User Guide[1]

DRAFT

Hilaire Fernandes[2]

January 15, 2014

[1] The electronic version of this book can be found at `http://drgeo.eu`
[2] hilaire@drgeo.eu

DRAFT

This document is published under Free Document License.
Author : Hilaire Fernandes
Peer review :

Contents

DRAFT

DRAFT

Introduction

Preliminary

Dr. Geo permits its users to create geometric figures, called sketches within Dr. Geo, and to manipulate them interactively, observing assigned geometric constraints. It also offers a gentle path into programming. Dr. Geo is usable in teaching situations from the primary to advanced levels. Its user interface is designed as a harmonious combination of simplicity and ease of use with advanced capabilities.

The Dr. Geo user interface, under the form of great simplicity, allows the beginner to get acquainted very quickly with the basic functions of the software. Then, as the user progresses, the more advanced aspects of the interface and the capabiities of Dr. Geowill become apparent, such as multiple methods of construction of each kind of object,[3] macro construction, multiple recording, scripts, Smalltalk sketches, and inheritance of Smalltalk in Dr. Geo. These advanced features generate little overload on the interface, which is why Dr. Geo is very pleasant for use in primary education; however they also make it very interesting for use in high school and university.

In the following sections, we explain the basic tools. Then the advanced features are presented in detail. We begin with a blank sketch. Open Dr. Geo, revealing a menu bar, a tool bar, and the welcome message ”Welcome to GNU Dr. Geo-Free interactive geometry by OFSET”. Click the new sketch button at the left of the toolbar.

The layout of the interface is as follows:

1. A *menu bar* with: `File – Edit – Points – Lines – Transformations – Numerics – Animate – macro construction`

2. An *editing toolbar* New figure; open and save figures; undo last action; redo last action; display grid; snap to grid; mode toggle for single or multiple object creation; select and move an object; erase an object; modify the style of an object; modify object properties.

3. A *construction toolbar*, which groups the tools for constructing objects in several tabs. It has the same functions as corresponding items on the menu bar.

4. In the right bottom corner, two wheels for moving the sketch horizontally and vertically.

5. In the right top corner, a wheel for scaling the sketch.

To create a new sketch, the user selects the New command in the File menu. We will indicate such menu choices in the form `File->New` from here on. For each new sketch,

[3] A particular command can create an object from different parameters. For example with the command for constructing a circle, the user can specify its centre and a point on the circumference or the length of its radius, among other options. Of course this command is represented only by one button, with Dr. Geo anticipating the construction intended by the user from the context. The immediate effect is thus a decrease of the cognitive load in the user interface, while offering a significant expansion of capabilities.

DRAFT

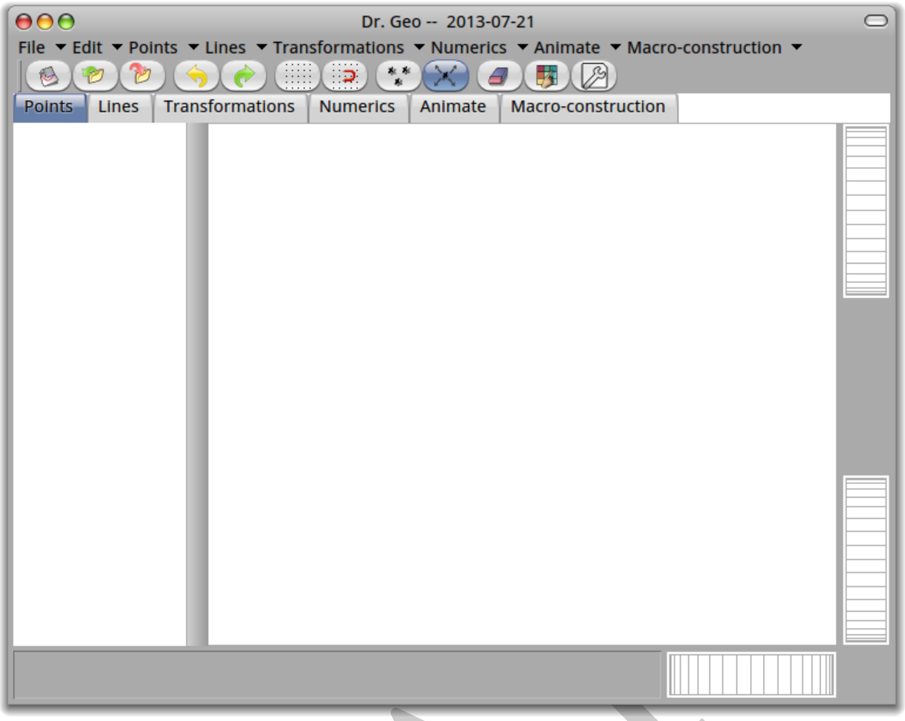


Figure 1: Dr. Geo window with a blank figure

DRAFT

a distinct window opens with its own menu and toolbar. The user can then create points, lines, circles, and so on, and control their properties.

Philosophy of the application

Dr. Geo II is Free Software[4] for interactive geometry on multiple platforms. It is a complete rewrite of Dr. Geo 1.1 in Pharo Smalltalk – `http://pharo-project.org`.

Dr. Geo 1.1 was written in C++ and integrated with a Scheme interpreter for writing scripts to make interactive programmed sketches. Dr. Geo II also provides the integration of scripts in geometric sketches so that interactive sketches can be written in a programming language.

The rewrite in Smalltalk was motivated by its unique dynamic qualities; it in fact allows us to push very far in our investigations of the interactive possibilities between the user and the application. So Dr. Geo is not only a convivial interactive geometry application but it is also, as distributed, a complete programming environment to study, to modify and to improve it.

For get an idea of these capabilities, the user is invited to click in the background of Dr. Geo – outside any window – and to press the key combination control-b. choose `Tools>System Browser` from the menu. The system browser that appears lets the user explore and modify Dr. Geo source code while it is running.

This access to the source code is part of the Dr. Geo DNA; it is part of the free software philosophy as well, for a completely open approach, unlocked, with great opportunities for learners. Far be it from us to pretend that Dr. Geois enough for building mental capacity, but it certainly can help.

[4]Free Software is software under a Free license such as GPL, which requires that its source code be made available for study, modification, and redistribution under the same license.

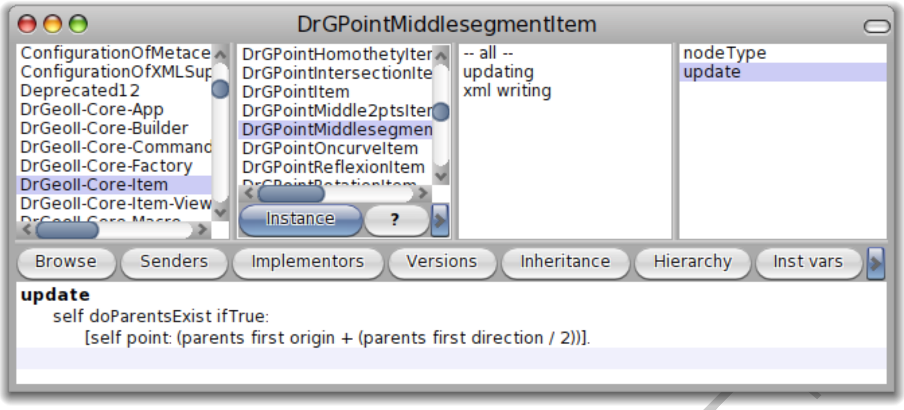


Figure 2: System browser opened on the Dr. Geo source code from Dr. Geo itself

DRAFT

With this same spirit of openness, programmed sketches and scripts – presented in the advanced tools section – rely on an advanced set of developer tools: browser, debugger, inspector, and more.

In the following we will not distinguish between the application names Dr. Geo II or Dr. Geo.

Dr. Geo on the web

Dr. Geo has its own web site at `http://drgeo.eu`. Here you will find the following information:

- how to get Dr. Geo,
- software documentation,
- information about the Dr. Geo project,
- reference material on applications of Dr. Geoin teaching.

DRAFT

Part I

Getting started

DRAFT

DRAFT

Chapter 1

Basic functionality

This chapter describes the tools used to construct a geometric sketch.

1.1 Tools for construction

These tools are ordered on six tabs. Clicking any of these tabs brings up an appropriate toolbar.

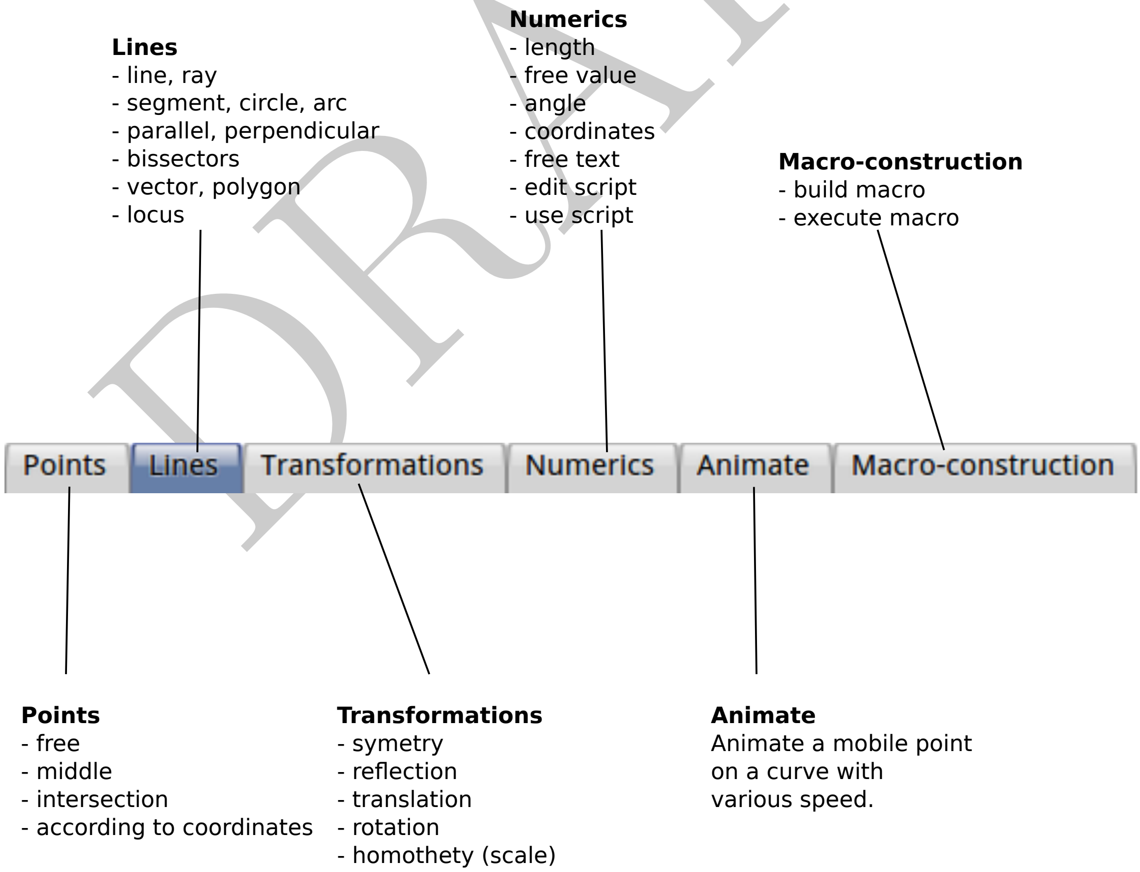


Figure 1.1: Tool categories and the descriptions

Dr. Geodefaults to a functional mode of operation. The user must click on a function

DRAFT

tab and then a function button, or select a function from a menu, before selecting the object or objects to apply it to, and executing the function once. At this point, the application returns to its default state, in which the user can select and move objects. The user can abort any function before making the final selection with the Select and move button on the toolbar, or the `Edit->Select and Move` command.

This is in contrast to a modal application, in which clicking a function button puts the application into a mode that allows the command to be used multiple times in succession. The commands for editing object styles and properties are always modal, allowing any number of objects to be modified without having to select the function repeatedly to apply it to each one. The user has to explicitly exit them, for example by returning to the select and move mode or starting some other command.

There is a mode toggle button for single/multiple creation, which switches from this functional mode to this modal mode, in which function selection is persistent. That is, selecting the line function allows the user to create multiple lines without having to select that function over again for each one.

When the user clicks one of these tabs, an additional toolbar displays immediately. It groups functions of the same family.

From left to right, we have access to tools to build points and lines, use transformations, calculate values, animate sketches, and manage macro construction and execution.

These functions also appear in the menus at the top of every Dr. Geo window.

1.1.1 Point tools

Free point

Create a free point in the plane or on a line object (segment, ray, straight line, arc, circle, locus):

6. In the first case, the created point can be moved anywhere in the plane of the sketch. To create it the user simply clicks in the background.

7. In the second case, the point can move freely only in the line where it was created; it is stuck on the object. To create this type of point, the user clicks on a line (i.e. straight line, ray, segment, circle, arc, etc.).

Middle

Create the middle point between two points, or the midpoint of a segment:

8. In the first case, the user selects two points.

9. In the second case, the user selects a segment.

To create one **intersection point** between two lines (i.e. straight line, ray, segment, arc, circle), select `Points->Intersection` , then the two lines in turn. Dr. Geowill throw an error if the user tries to find an interection with a locus.

It is enough to click in the intersection of these. Moreover, Dr. Geoindicates it by the help bubble: *Intersection*.

Intersection

Create one or several intersection(s) of two lines (i.e. straight line, ray, segment, arc, circle). The user has to select two lines. When one or both of the chosen lines is an arc or a circle then two intersection points may be created.

DRAFT

Point defined by coordinates

Coordinates

How to place a point by specifying its coordinates A possibility is to place two free values in the sketch – tool Numerics, section 1.1.4, p. 8 – then build the point with these two values as coordinates – tool point defined by its coordinates, section 1.1.1, p. 5. This possibility has an advantage over the previous one for some purposes. The point so constructed cannot be directly moved with the mouse. The point is constrained in its position.

Create a point defined by its coordinates. The user selects a script

returning a pair of x-y coordinates – section 4.1.4, p. 34 – or two values:

the first one is the abscissa (the x coordinate) and second one the ordinate (the y coordinate).

How to construct a point constrained by its coordinates? This function is mainly used to construct a locus of point.

This construction supposes beforehand the existence of two values – section 1.1.4, p. 8 – the point is then constructed by selecting these two values.

1.1.2 Line tools

Line

Line

Create a straight line defined by two points. The user clicks on two points, or clicks on one point and drags to another.

Parallel line

Parallel

Create a line parallel to a given direction and going through a point. The user clicks a point and a direction (i.e. straight line, ray, segment or vector).

Perpendicular line

Perpendicular

Create a line perpendicular to a given direction and going through a point. The user clicks a point and a direction (i.e. straight line, ray, segment or vector).

Perpendicular bisector

Perpendicular bisector

Create a perpendicular bisector given a segment or two points. The user clicks on a segment or two points, or clicks and drags.

Angle bisector

Angle bisector

Create the angle bisector of an angle formed by three points. The users clicks on a geometric angle (defined by three points)

or clicks on three points. The line will bisect the angle at the second point.

DRAFT

Ray

Ray

Create a ray defined by two points. The user clicks on two points. The first one is the origin, the second one a point on the ray.

Segment

Segment

Create a segment given two points.

Vector

Vector

Create a vector given two points. The user select two points, the origin, the second one the terminal point.

Once the vector is created, it can be moved independently of the two points. This remains true when the vector is built from a transformation – section 1.1.2, p. 7.

Circle

Circle

Create a circle. The user can create it from different selections:

10. the centre and a point on the circle,
11. the centre and a number (the radius),
12. the centre and a segment whose length is the radius.

Arc by trois points

Arc

Create an arc going through three points. The first one is the starting point, the third one is the end point, and the second one is a point on the arc.

Arc, centre

Arc (center)

Create an arc defined by its center and by starting and ending points. The first one is the centre, the second one is the starting point, and the third one is the end point.

Locus

Locus

Locus defined by two points. The user selects two points, one free on a line, the other one depending on it, so that moving the first point makes the location of the second point change. This dependent point is called a relative point.

For example, we can construct an ellipse based on its property that the sum of the distances from each point on the ellipse to two given points is constant.

13. Create free points O and A, which will be the given points.
14. With center O and radius greater than the length OA, create a circle.
15. Create a free point B on the circle.

DRAFT

16. Join the segments OA and OB.

17. Construct the point C, the midpoint of AB.

18. Create a perpendicular bisector to AB.

19. Construct the point D, the intersection of OB and the perpendicular bisector to AB.

20. Construct the locus with free point B and relative point D.

Since CD is the perpendicular bisector of AB, the triangles ACD and BCD are congruent (side-angle-side). Therefore AD = BD. Then AD + DO = BD + DO, which is the radius BO, which is constant. D lies on the ellipse. Moving B all around the circle moves D all around the ellipse.

Polygon

Polygon

Create a polygon defined by n points. The user selects n+1 points defining the polygon submits. The first and last ones are the same point, it indicates to Dr. Geo the selection is over. Mobile point can be added on polygon, but it is not possible to compute an intersection with another line. However geometric transformation of polygon are allowed.

Polygon regular

Regular polygon

Create a regular polygon defined by two points and a numeric value. The user selects its center, one vertex, and a value indicating the number of vertices. If the value selected is not an integer, it is truncated to the next lower integer.

1.1.3 Transformation tools

Reflection

Reflection

Transform an object by a reflection. The user clicks on the object to transform and the axis – straight line – of the reflection. When the object to transform is a straight line, the first clicked is the axe of the reflection.

Symmetry

Symmetry

Transform an object by a central symmetry. The user clicks on the object to transform and the centre of symmetry (a point). When the object to transform is a point, the first clicked is the centre of the symmetry.

Translation

Translation

Transform an object by a translation. The user clicks on the object to transform and a vector. When the object to transform is a vector, the first clicked is the vector of translation.

DRAFT

Rotation

Rotation

Transform an object by a rotation. The user clicks on the object to rotate, a centre (point) and an angle. When the object to transform is a point, the first clicked is the centre.

The rotation angle can be defined by three different means:

- **numeric value**: the angle is therefore the value interpreted in radians (Examples of numeric value: free value, distance between two points, segment length, a coordinate, a value returned by a Dr. Geo script, etc.),
- **a geometric angle defined by three points**: its measure is displayed in degrees. Beware, in this case the measure ranges only in the interval [0 ; 180]. The angle has to be put in numeric form using the angle function described below.
- **an oriented angle between two vectors**: its measure is displayed in degrees in the interval]-180 ; 180]. The angle has to be put in numeric form using the angle function described below.

Homothety (scale)

Homothety (scale)

Transform an object by scaling it. The user clicks on the object to transform, the centre and the factor (i.e. a number). When the object to transform is a point, the first clicked is the centre.

1.1.4 Numerics and text tools

Distance, length & value

Distance, length, value

Create a numeric value. The value, depending on the user selection, may be computed or user edited:

21. for two points it is the distance between these two,
22. for a segment it is its length,
23. for a vector it is its norm,
24. for a circle it is its perimeter,
25. for an arc it is its length,
26. for a line it is its slope,
27. for a point and a line it is the distance between these two,
28. a **mouse click on the background** creates a free value, initially 0, that the user can edit.

A newly created free value can be edited by clicking it. Later on, it can be edited by clicking the property button on the toolbar, described below, and then selecting the value.

This last possibility is very interesting for some situations. It lets the user set an arbitrary value to use for a given length, the radius of a circle, an angle measure (in radians) or the coordinates of a point. The value is then used by other specific tools to construct a circle, a rotation or a point defined by its coordinates.

DRAFT

Angle

Angle

Compute the angle defined by three points or two vectors. In the first case, the angle is considered as non oriented (i.e. angle whose measure belongs to the interval [0 ; 180]). In the second case, the angle is oriented and its measure belongs to the interval]-180 ; 180].

Coordinates

Coordinates, equation

Create the coordinates (abscissa and ordinate) of a point or vector.

Free text

Text

Add a block of text in the sketch. The user clicks at the desired position, then edits the text. To edit the text again, the user clicks the property button described below, then the text.

Inserting a Smalltalk script

Use a script

Insert a Dr. Geo Smalltalk script in the sketch. The script expects a number of objects as arguments, and requires a target to attach the output to. It returns an object whose text representation is printed in the sketch, at the position clicked by the user. A script can be used for its side effects or for the value it returns, ready to be used for subsequent constructions. See the Smalltalk Script chapter for more details.

Editing a Smalltalk script

Edit a script

Edit or create a Smalltalk script. A script editor is opened for the user to edit any existing script or to create a new script.

The Dr. Geo Smalltalk scripting system is reviewed in detail in the chapter on advanced functionalities II, specifically in the section 4, p. 27.

1.1.5 macro construction tools

Create a macro construction

Build macro

Extract a construction sequence from the sketch and compile it as a macro construction. macro constructions are covered in detail in the section 3, p. 21.

[

Play a macro construction

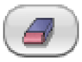
Execute Macro

Execute a macro construction. The macro construction can be newly created or loaded from a file.

DRAFT

1.1.6 Misc tools

Delete an object



An object can be deleted using the eraser tool or the `Edit-->Eraser` command. Objects that depend on the deleted object will also disappear. When deleting a curve, the points used to define it remain behind. The action is undoable with the undo button in the toolbar or the undo menu item in the Edit menu.

Change the style of an object

Each geometric object has its own style attributes, which can include colour, line style, thickness, label, size and shape, and properties such as Fill, Translucent, and Hidden. An object that is hidden remains in the sketch, but is not seen except when editing properties, so that it can be found. Hiding intermediate construction is often useful to avoid clutter in the sketch. All these attributes are edited from a side panel displayed when the user enters style mode. To do so, click the  button from the toolbar, and then select a particular item in the sketch.

Style of point. The lateral panel for style of point is for points within every kind of object. Name, Colour, shape (x, dot, or block), size, visibility, and locking are adjusted from there. Only the position of free points on the plane or on a curve is lockable.

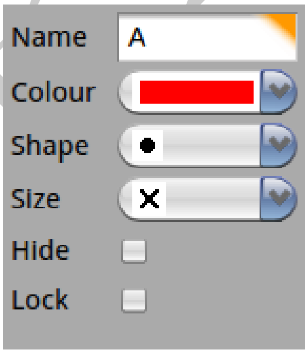


Figure 1.2: Side panel to edit style of point

Style of line. The side panel for style of line is for straight lines, rays, segments, vectors, circles, arcs, loci, and polygons. Name, colour, thickness, solid or dashed styles, and visibility are adjusted from there. Some properties only apply to particular kinds of line, as in the case of marks and arrowheads on segments.

Style of value. The lateral panel for style of value is for all kinds of numeric value: free value edited by the user, measure, computed results from a Dr. Geo Smalltalk script. Colour, label and visibility are adjusted from here. The value can also be locked.

DRAFT

Figure 1.3: Lateral panel to edit style of line

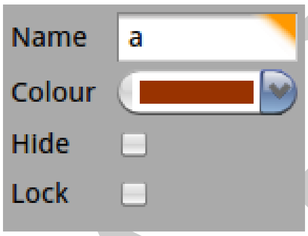


Figure 1.4: Lateral panel to edit style of line

Object property

Some objects' numeric or text values are editable. This includes the coordinates of free points on the plane or on a curve, free values and scripts. To edit the property, select this tool, then choose one of these objects in the sketch. A dialog box will appear:

Free point. Selecting a free point, a dialog box let you edit its abscissa and ordinate.

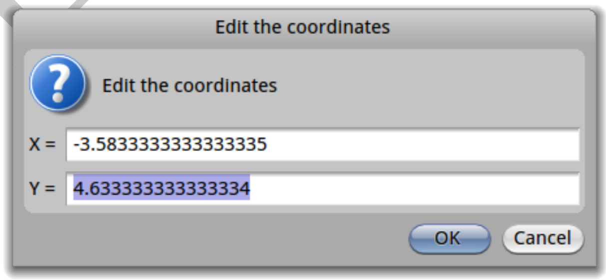


Figure 1.5: Edit the coordinates of a free point

Free point on curve. Selecting a free point on curve, a dialog box let you edit its curvilinear abscissa. It is normalised in the interval [0 ; 1]. This allows a curve to be treated as a parametric curve.

DRAFT

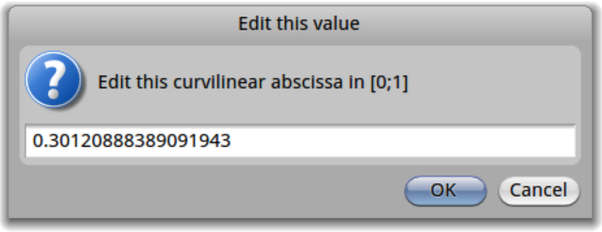


Figure 1.6: Edit the curvilinear abscissa of free point on a curve

Free value. Selecting a free value, a dialog box let you edit its value.

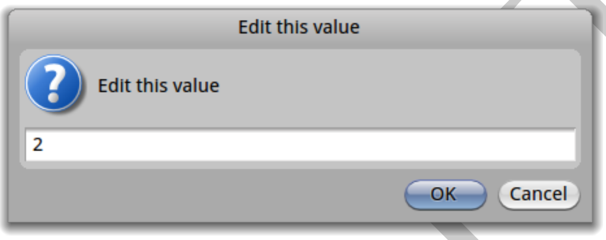


Figure 1.7: Edit a free value

Script. Selecting the value displayed from a script, an editor is opened to let you study and edit it. To save any modification to a script, use the key shortcut CTRL-S or the entry *Do-it* in the contextual menu over the script (right mouse click to display it).

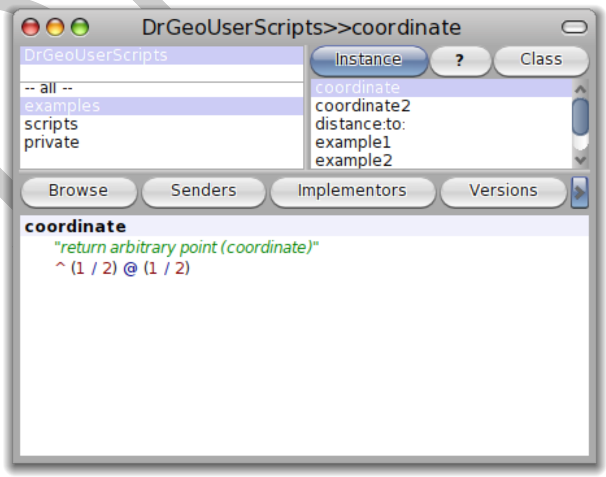


Figure 1.8: Edit a script

Text. Selecting a script, a dialog box let you edit it. The text can be formatted in several lines with carriage return – keyboard key RETURN.

DRAFT

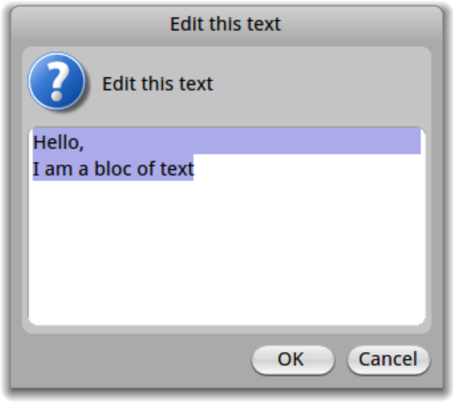


Figure 1.9: Edit a text

1.2 Misc functions

1.2.1 Moving the Sketch

The sketch can be moved with the wheels in the right bottom corner or directly with the right mouse button.

1.2.2 Scaling the sketch

The sketch scale can be adjusted with the wheel on the top right of the window. The mouse wheel offers this same service; press simultaneously the keyboard key SHIFT to speed up the scaling.

1.2.3 Moving an object

An object is moved by clicking and dragging with the mouse. The whole sketch is then recomputed and redrawn to respect the constraints specified. Almost all geometric objects can be moved. If necessary, Dr. Geo moves the associated free points. For example, when the user drags a line defined by two points, the points are moved simultaneously.

In this mode, it is possible to change the nature of a point from:

- a free point on the plane
- a free point on a curve
- an intersection point

to a point as:

- a free point on the plane
- a free point on a curve
- an intersection point

For example, transforming a free point on the plane into an intersection point. However, there is one constraint: it is not possible to mutate a point toward a "younger" line (as free on the line or intersection with the line). Younger line means created chronologically after the point.

DRAFT

Pressing the keyboard key SHIFT while grabbing a point will display a balloon help text about the point you can mutate and the possible destinations.

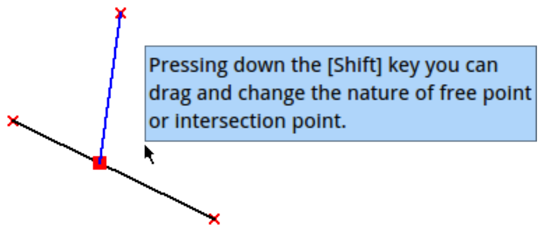


Figure 1.10: Pressing [Shift] to mutate a point

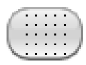
1.2.4 Merging points

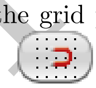
Two free points can be merged: to do so drag one point over the second destination point and hold for a few seconds.

1.2.5 Cloning curve

Various type of curve can be cloned to quickly duplicate identical objects. To do so press and hold the mouse button over the curve to clone. The curve will blink, and then a cloned curve is displayed, slightly offset from the original, ready to be positioned as the user chooses.

1.2.6 Grid

It is possible to show or hide a grid in every sketch with the  button in the toolbar. The scale of the grid adjusts itself when the sketch is enlarged or shrunk. When saving a sketch, the grid state is saved as well.

When creating or moving points, it is possible to constrain them to snap to the grid, so that the point will appear at the grid position closest to the mouse pointer. To do so, activate the magnet tool from the  button in the toolbar.

DRAFT

Chapter 2

Files and documents

Sketches are saved in two different ways, one sketch per file or a set of sketches and macro constructions in one file (i.e. a Dr. Geosession). We recall that documents are saved with the extension **.fgeo**. When providing the file name, you don't need to write this extension. Dr. Geo will do it for you.

2.1 Rename a sketch

From the menu `File->Change title`, the user changes the name and the title of the active window. This function is useful when the user saves a set of sketches in a single session file.

2.2 Save a sketch

From the menu `File->Save`, the sketch of the active window is saved. The user is prompted for a name.

> **(!)** Dr. Geo can work with several sketches at the same time. The user switches from one sketch to another one with the task bar in the bottom of the Dr. Geo environment.

With the menu `File->Save as...`, the user saves the document under another name.

The documents are saved in the Dr. Geo application structure, in the folder `DrGeo.app/MySketches`. To save at an arbitrary location in the hard disk structure, use the menu `File->Save at...`

> **(!)** When saving a document, a network option is available in the dialog box. If selecting this option, the user must also provide a network share name – arbitrarily chosen by the user – and the document name. This option lets a group of networked users share sketches

2.3 Export a sketch

The menu `File->Export as bitmap` exports the sketch of the active window in a picture file – bitmap – in the standard format PNG[1].

The pictures are saved in the Dr. Geo structure application, in the folder `DrGeo.app/MyExports`.

[1] `http://www.w3.org/Graphics/PNG`

DRAFT

2.4 Save a session

A session is a set of Dr. Geo documents – sketches and macro constructions – the user saves in **one** file. It eases the preparation of pedagogical sequences.

From the menu `File->Save multiple` , the users accesses a dialog box to select the document to integrate in the session file.

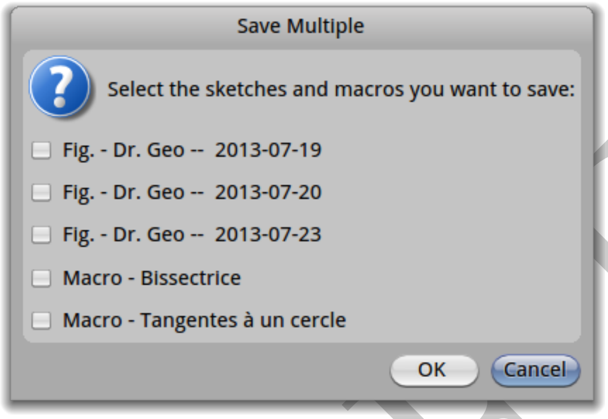


Figure 2.1: The Dr. GEOsession dialog box

In this dialog, the list of all active documents is presented in column. Each line is prefixed with a tag `Fig.` or `Macro` according to the document type, the second part is the name of the document.

(!) As for now, a session knows about two types of document: 2D interactive sketch and macro construction.

The user selects the documents to integrate in the session then provides a name for the composite file, and clicks `OK` .

(!) The menu `File->Save multiple` is the only method for saving a macro construction in a file.

2.5 Save a macro construction

To save one or several macro constructions in a file, proceed the same way as for saving a session – multiple save. From the dialog box to save a session, select one or several macro construction to save, then input the file name. That's all.

Therefore, it is possible to build macro construction libraries, one per file or several per file following a given theme.

2.6 Open a file

Whenever the user saved an individual sketch or a session in a file, the procedure to open it is the same from the menu `File->Open` .

If the session contains macro constructions, they are automatically added to the macro menus and player, ready to use in any existing Dr. Geo window.

DRAFT

Part II

Advanced functionalities

DRAFT

DRAFT

Introduction

In this part, we present features to extend Dr. Geoś capabilities, or to adapt it to a given pedagogical situation.

The first feature is the *macro construction*. It lets the user record a sequence of construction steps in a single command. Once recorded, it can be replayed and saved in a file to be opened later in another sketch.

The Dr. Geo Smalltalk script is another feature to extend Dr. Geo behaviour. A script appears as a sketch item like any other. As input, it expects zero, one or more references to geometric items from the sketch and it returns a reference whose print representation is placed in the sketch. It is in fact a programmed method [2] placed in a sketch, and evaluated each time the whole sketch is updated (i.e. when the sketch need to be redrawn). A Dr. Geo Smalltalk script may be useful for its returned object or for its side effects, depending on the wishes of the user.

Going a step further, Dr. Geo offers Smalltalk sketches. In this case the whole interactive geometry sketch is described in Smalltalk source code. Its interest is the functional approach to describing a sketch[3] compared to the linear, declarative method of constructing a sketch with the mouse.

DRAFT

[2]Comparable to a procedure for users of other programming languages.
[3]For example recursively

DRAFT

Chapter 3

Macro Construction

A macro construction is similar to a programmed procedure or function, receiving as input items from the sketch and providing as outputs another set of items in the sketch. A macro is built according to a model defined by the user. That means that the user first needs to do the complete construction sequence in the sketch, then ask to Dr. Geo to record it in a macro construction. The macro construction can then be replayed and saved in a file like any sketch.

To record a construction sequence, Dr. Geo needs to know the initial items of the sequence and the final – output – items. Of course, the final items must depend *only* on the input items, otherwise Dr. Geo cannot deduce the final items from the initial ones.

Thus, Dr. Geo traces the logic of the construction sequence for the specified outputs, and records it in a macro construction. Then, the user can repeat this sequence. When playing the macro construction, it only asks for the initial items – of the types specified – in the sketch and constructs the resulting – output – items.

(!) Intermediate invisible items are constructed by the macro construction. They are necessary to reproduce to complete the construction sequence and the resulting output items.

To illustrate the use of a macro construction, we take the example of a circle going through three points.

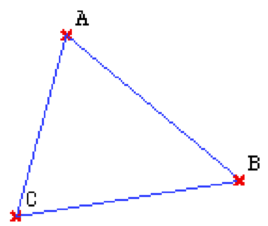


Figure 3.1: The initial sketch

Before creating the macro construction, the user has to construct the final sketch to serve as a model, as shown in the figure below.

3.1 Creating a macro construction

The user now tells Dr. Geo he wants to define a macro construction from this sequence.

DRAFT

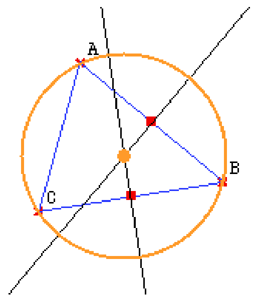


Figure 3.2: The sketch with the final construction

He activates the function `Build a macro` from the toolbar or from the menu `macro construction>Build macro`.

In the newly displayed dialog box, the user selects the input and output objects, and enters a name and description of the macro construction.

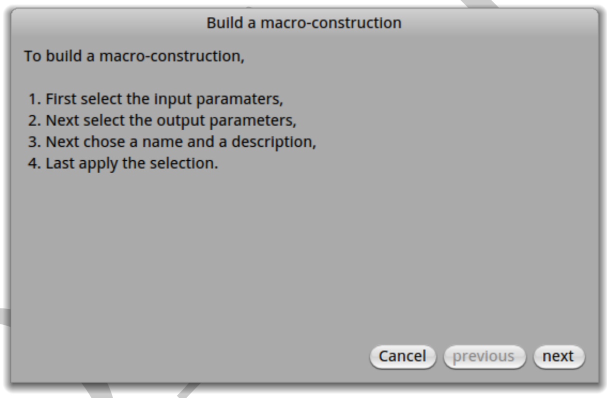


Figure 3.3: Introduction page in the dialog box to construct a macro construction

The second page in the dialog box lets the user select the input objects. In our example, it is the three initial points. The user just needs to go to this second page and then click on the three points A, B and C in the sketch. The selected points blink and the names are displayed in the dialog box.

In the third page, the user selects the output objects. In our example, we want the circle and its centre as the result of the macro construction. The user clicks on these two objects in the sketch. When selected they blink, and their names are displayed in the dialog box.

In the fourth page, the user inputs a name and a description for the macro construction. This information is displayed when the user plays a macro construction. It helps to disambiguate macro constructions.

Next, the user validates the definition of the macro construction by pressing the `Apply` button and examining the results. He can also step back to the previous steps to adjust the parameter objects.

DRAFT

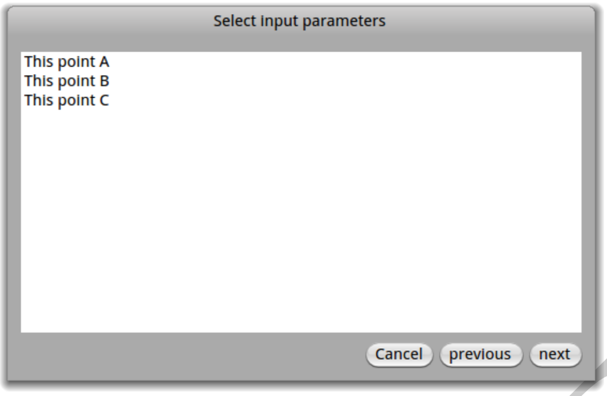


Figure 3.4: The second page, with the three points selected

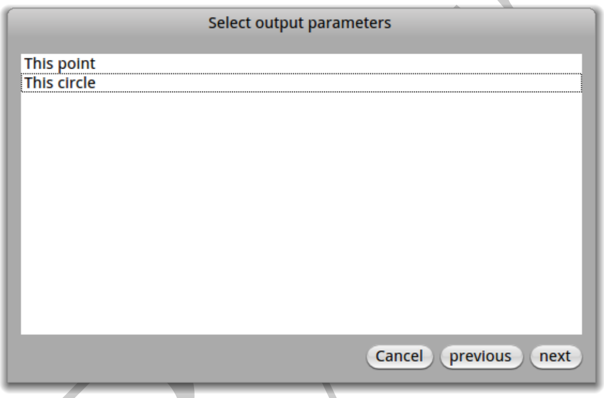


Figure 3.5: The third page, with the circle and the centre selected

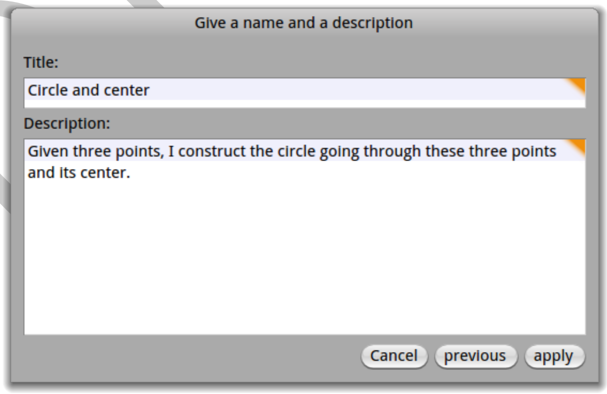


Figure 3.6: The fourth page, with the name and description of the macro construction

(!) If the selections of the input and output objects do not match (Dr. Geo cannot extract a construction sequence, because an output depends on an object not selected as an input), the macro construction cannot be created. In this case the user needs to reconsider the input and output parameters. He or she can step back to the second or third page in the dialog box to adjust the choices.

DRAFT

A this stage, the macro construction is created and recorded in Dr. Geo. In the next section, we will see how to use it.

3.2 Play a macro construction

3.2.1 With the dialog box

To execute a macro construction, the user clicks on the Execute Macro button in the toolbar or selects the menu `macro construction>Execute macro` in the sketch window. A dialog box describing the procedure appears.

On the second page of this dialog box the user selects the macro construction in the upper list, and then the input parameters in the sketch. Once all are selected, the macro construction is executed automatically and the output parameters are built and displayed in the sketch.

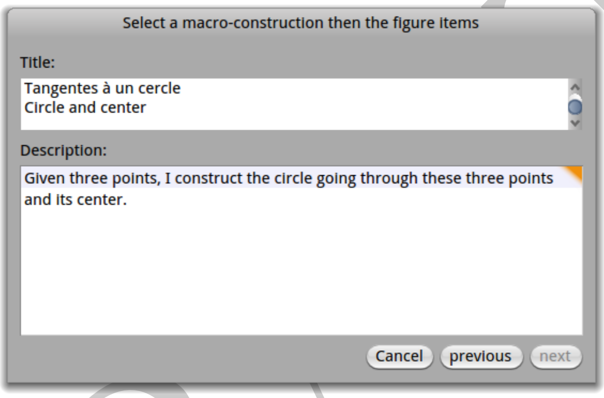


Figure 3.7: The user selects the input parameters in the sketch

In our example, the macro construction expects three input parameters (three points) and constructs a circle and its centre. Therefore, to execute it, you need a sketch with at least three points.

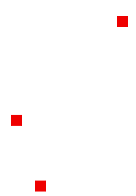


Figure 3.8: Sketch with three points

Once we feed three points to the macro we get the expected circle and centre.

3.2.2 With the macro construction menu

The macro constructions loaded in memory are listed in the window menu `macro construction`. Selecting a macro construction from this menu let you execute it directly. After choosing one in the menu, select the input parameters in the sketch. The macro construction is executed as soon as all the input parameters are selected.

It is a dialog-less mode of execution.

DRAFT

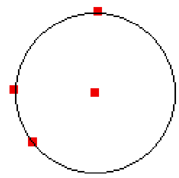


Figure 3.9: A final sketch with the circle and its centre

As described in the Files and documents chapter, to save a macro or collection of macros to a single file, select `File-> Save multiple` , and then select the items to include in the saved file. The same process works for saving a sketch with multiple macros.

To add a macro from a saved file to a sketch, create or open the sketch and open a file containing just the macro.

DRAFT

DRAFT

Chapter 4

Dr. Geo Smalltalk script

DRAFT

Dr. Geo is a dynamic application written in Pharo Smalltalk. This means that it is possible to modify it within itself as it is running. We have exploited this feature with the `Numerics->Use a script` command to define sketch items in Dr. Geo which are in fact Smalltalk scripts – code snippets – to extend the possibilities of Dr. Geodynamically, without limit. But what exactly is Smalltalk?

> *Smalltalk is an object-oriented, dynamically typed, reflective programming language. Smalltalk was created as the language to underpin the "new world" of computing exemplified by "human–computer symbiosis." It was designed and created in part for educational use, more so for constructionist learning, at the Learning Research Group (LRG) of Xerox Palo Alto Research Center (PARC) by Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Scott Wallace, and others during the 1970s. The language was first generally released as Smalltalk-80. Smalltalk-like languages are in continuing active development, and have gathered loyal communities of users around them. ANSI Smalltalk was ratified in 1998 and represents the standard version of Smalltalk. .*[1]

In this definition, object-oriented means that everything in Smalltalk is an object defined in a class, with its behavior determined by methods defined in that class. This is in contrast with the notion of a variable associated with a data type in most programming languages, where the type system sets few limits on what can be done with values.

In statically typed languages, the type of a variable has to be declared before it can be used, and that type cannot be changed within the program. In particular, the sizes of arrays and other composite data structures must be declared in advance. In dynamically typed languages, a variable name can be repurposed to refer to an object of a different class with a simple assignment, items can be added to or removed from data structures at will, and the class definition can be changed while a program is running.

Reflection in programming languages refers to the ability to examine and modify properties of objects within the live system, rather than having to read the source code in which variables are declared to determine their types and infer their properties. Reflection tools in Smalltalk include the system browser, the inspector, the object explorer, and the method finder.

This abstract from the book preface *Pharo By Example*[2] describes the Smalltalk environment used by Dr. Geo:

> *Pharo is a modern Open Source development environment for the classic Smalltalk-80 programming language.*

[1] Wikipedia Smalltalk article (`http://en.wikipedia.org/wiki/Smalltalk`). Page view the 24 Jully 2013. Contents under the license CC-BY-SA 3.0 (`http://creativecommons.org/licenses/by-sa/3.0/deed.fr`).

[2] `http://pharobyexample.org`

> *Pharo strives to offer a lean, Open Source platform for professional software development, and a robust and stable platform for research and development into dynamic languages and environments.*

Dr. Geo uses the Smalltalk environment to create a comfortable environment for writing scripts providing access to the programming interfaces for geometric objects. These interfaces are the set of methods in the definitions of the types of the objects.

Thus the user can write scripts to manipulate the sketch items, and as scripts pinned in the sketch are also sketch items, they do not need to be in separate files, but can be saved in the sketch file.

Scripts are not edited as sketch items but as methods stored globally in Smalltalk. Using a script and pinning the result in a sketch makes it part of the sketch so that it can be saved and reloaded with the sketch. However, there is no way to keep scripts in different sketches separate if they are open at the same time. This is a design bug, because users will often have all of their work polluted with scripts meant to be used in other sketches, unless they exercise great care to avoid this. In fact, the default state of Dr. Geocontains a number of useless scripts that could have been put into the examples category or even omitted, so that any new session would start with an empty scripts category.

A workaround is to close Dr. Geoand open a new session after working with a sketch containing multiple scripts and before creating any new sketch, which can then begin without any extraneous scripts.

While it is possible to put scripts for a sketch into a new category for creating them, it is not possible to select by category when using a script. Furthermore, in the use a script dialog, scripts seem to be offered in random order. It is certainly not alphabetical or temporal order.

Scripts in Dr. Geoare executed in the Pharo Smalltalk environment. The user thus benefits from the excellent developer tools of the environment: class browser, inspector, debugger, etc. The user wishing to explore the power of scripts is invited to study the book *Pharo By Example*, where he or she will learn the Smalltalk language and its environment.

For our purposes in this chapter, it is necessary mainly to understand how to use a Workspace. Open a new Workspace by clicking in the background and selecting `Tools->Workspace` . The workspace provides

- Basic text editing capability
- Syntax highlighting
- Context-sensitive selector (method name) completion
- Access to a menu of editing commands and development tools

Text editing capabilities in a Workspace consist of the usual Cut, Copy, and Paste; Find and Replace; Undo and Do again. An Extended search function displays information about selected items using various development tools. There are also Accept and Cancel functions. Typing in the Workspace causes a highlight to appear at the upper right of the window. Accept checkpoints the contents of the window, empties the undo stack, and causes the highlight to disappear. Editing and then cancelling takes the window back to the last checkpoint without having to undo each action individually.

When editing, object names appear in blue, method names and symbols such as := in black, and unrecognized text in red. This includes incomplete names, undeclared local variables, and also syntax errors. When the user is typing a selector, a menu appears showing selectors starting with the typed text that are usable in the current context, that is, for the object that will receive this message. Move up or down in the menu with the arrow keys, and accept the highlighted selector by pressing the tab key. If the selector has more than one part, all of the parts are entered into the workspace.

Development tools provided on the workspace menu, accessed by right-clicking in the workspace, are

DRAFT

- **Do it** Execute the selected text or the current line otherwise
- **Print it** Execute text and display the result
- **Inspect it** Open an Inspector on a selected class name, or execute selected text and open an Inspector on the results.
- **Explore it** Open on object Explorer on a selected class name, or execute selected text and open an Explorer on the results.
- **Debug it** Open a Debugger on selected code
- **Profile it** Open a Profiler on selected code

Development tools accessible in the Extended search menu are the following.

- **browse it**
- **senders of it**
- **references to it**
- **selectors containing it**
- **method strings with it**
- **method source with it**
- **class names containing it**
- **class comments with it**
- **change sets with it**

Each one opens the tool specified on the text selected or on an object that that text names, if any. If there is none, a message may appear saying so.

Some of the functions of these development tools can be discovered through exploration, clicking on every button to see what happens. For further details see Pharo Smalltalk documentation.

The Workspace menu

4.1 Script by the example

There are two phases to using scripts:

- creating the script
- inserting the script in a sketch one or more times with different parameters

The tool to create or edit a script is on the menu `Numeric->Edit a script` . It is also reachable from the toolbar.

The tool to use a script is on the menu `Numeric->Use a script` . It is also reachable from the toolbar.

A script can be defined to receive 0 or more arguments (input parameters). After selecting a script to insert in the sketch, the user clicks on the items used as arguments. Then a final click somewhere in the background sets the location for the returned value.

In the following section, we present a few script examples so that the function and power of scripting will be more easily understood. The script and the macro construction give a

special dimension to Dr. Geo – with a different positioning[3] – to let the user go into areas that the software was not initially planned for.

It is important to understand that most Pharo Smalltalk resources are accessible in scripts. This is particularly true for the class and method libraries[4], which we will use a lot.

4.1.1 Script without parameter

First example: The procedure to create a script without an input parameter is as follows:

29. **Edit the script**

    (a) Select `Numeric->Edit a script` in the menu to open the script editor:

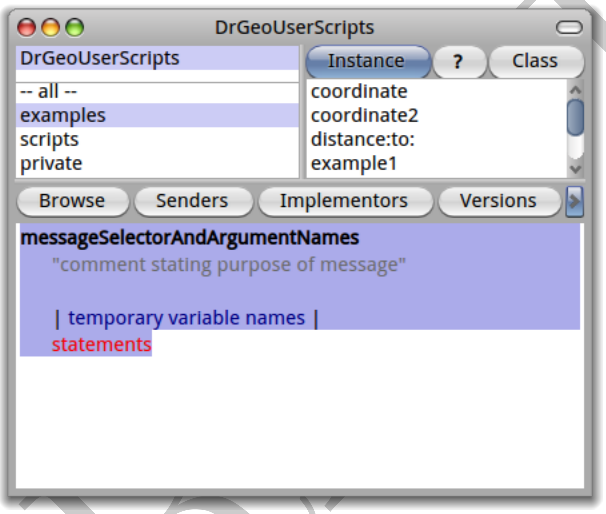


Figure 4.1: Script editor

Within the script editor window there are three panes:

- In the top left, the script categories to keep in good order: `examples`, `private`, and `scripts`. It is in this last one that you put your scripts.
- In the top right, the script names in the selected category. When selecting one, its source code is displayed in the editor pane at the bottom.
- In the bottom, the source code editor for the selected script. It is there you create or modify the scripts. To accept a modification, press the keyboard keys Ctrl-S.

    (b) In the script editor, select the category `scripts`. The text editor in the bottom will display a script source code template such as the one shown in figure 4.1.

    (c) Input the source code as below and as shown in figure 4.2 :

```
myFirstScript
"Hello, World is the traditional first program in any language.
It simply displays a text message."
   ^ 'Hello, World!'.
```

[3] A macro construction is geometry oriented whereas a script is numerically/computation oriented.
[4] For example, mathematical functions.

DRAFT

Save the script with Ctrl-s. Dr. Geo will ask your last name and first name to track the history of the script modifications. The first script line, `myFirstScript` is the name of the script, followed by the source code. It is followed by an optional comment of one or more lines between quotation marks. The comment should usually explain the purpose of the script, what are the expected parameters, and what are the conditions for its use. We strongly encourage you to document carefully to prevent confusion and error.

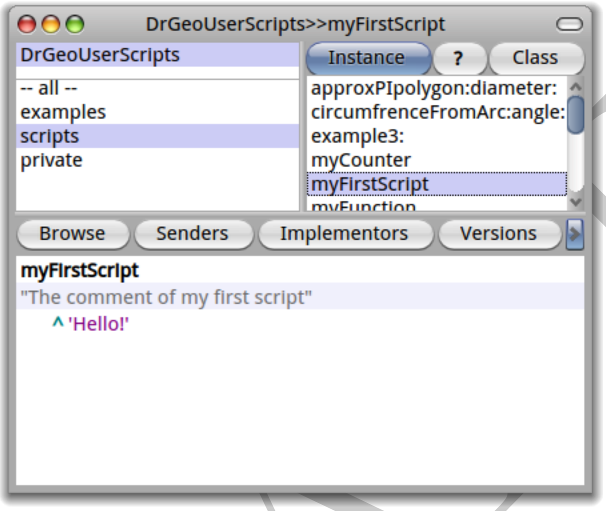


Figure 4.2: My first script

The script editor can now be closed.

30. **Using the script in the sketch**
Select `Numeric->Use a script` in the menu. In the displayed dialogue box, choose the script `myFirstScript`. Note: each time a script is selected, its descriptive comment is displayed in the bottom of the dialogue.

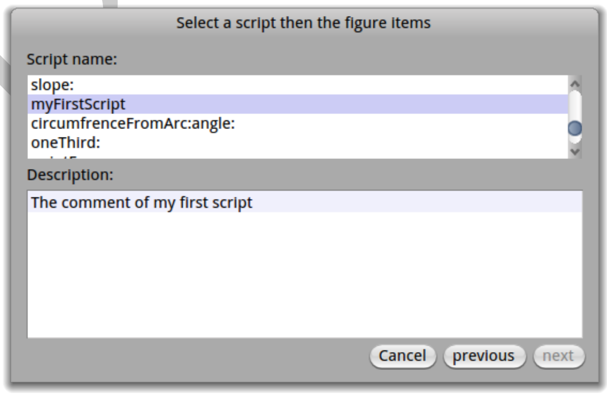


Figure 4.3: Select a script

Once the script selected, click on the sketch at the place to pin it. In this example the script only returns the message “ Hello, World ! ”. The returned value is always printed in the sketch. If necessary it is converted to a text representation.

DRAFT

In the following examples, we only give the source code of the Smalltalk scripts. To use them in Dr. Geo, just follow the steps explained previously.

A random number generator and more: If you need a simple random generator for numbers between 1 and 10, the following script exactly does that:

```
random
"I return a random number continuously updated"
   ^ 10 atRandom.
```

At each sketch update, it generates a different random number in the interval [0 ; 10]. Updates occur when the sketch is changed by dragging an object.

In case you prefer a random decimal number in the interval [0 ; 1], use this script:

```
random2
"I return a random decimal number continuously updated from 0 to 1"
   ^ (101 atRandom -1) / 100.
```

(!) Some details:

- The value returned by the script is the result of the expression after the ^ .
- The returned value can be of any class. Dr. Geo prints its text representation (a string).
- To return the value of a variable, it is enough to put its name after the symbol ^ .

Calculate common values To access an approximate value of π with the maximum precision available in Smalltalk:

```
pi
   ^ Float pi
```

This returns 3.141592653589793

or e :

```
e
   ^ Float e
```

This returns 2.718281828459045

These returned values are usable in the same way as any other value generated by Dr. Geo. Even for such small things, the script is your friend. But it can do much more interesting things when it receives input parameters.

Indeed, so far the scripts have had no arguments, so that it was not necessary to select items in the sketch when inserting the script in the construction. Of course the real interest in scripts comes from returning the values of numerical computation, pinned in the sketch for use with other constructions or scripts. In the following sections we show the use of scripts in a cascade.

4.1.2 Script with one input parameter

The procedure to create a script with one input parameter is mostly the same:

DRAFT

31. When editing the script (`Numeric->Edit a script`), the argument is inserted in the first line with the script name.
    For example to calculate the distance from the origin to a point, we write the following script:

    ```
    distanceToOrigin: item
    "Return the distance from origin to a point"
       ^ item point dist: 0@0
    ```

    A few explanations:

    - `item` is the argument of our script, which must be a point. It is in fact an instance of the class `DrGPointItem` [5]
    - `DrGPointItem` has an instance method `#point` returning its coordinates. Thus from the argument we can extract information, here its coordinates, and calculate with it.
    - `0@0` is an instance of the class `Point` with coordinates (0,0).
    - `#dist:` is a keyword message [6] from the class `Point` expecting as its unique argument another instance of `Point`. It calculates the distance between these two instances. It can be understood as: "distance between item point and (0,0)". The keyword message syntax is very specific to Smalltalk. The arguments are declared in the message name line of a method.

32. To use a script (`Numeric->Use a script`), Dr. Geo expects from the user to click on appropriate objects, then somewhere in the sketch.
    Attention: If an item other than a point is selected when a point is expected, Dr. Geo throws an error, opening a debugger window stating the problem and offering further information. The debugger window can be safely closed. To continue, select the script again.

Depending on the type of argument received by the script, various methods are available: to get its value, its coordinate, etc. A list of the methods you can use are presented in section 4.2, p. 36 below.

4.1.3 Script with two input parameters:

Let's say we want to calculate the distance between two points. To do so we create a script with two input parameters inserted on the name line of the script. We name it `distance:to:`, where each ":" indicates the place of the parameter. Therefore in the script editor we write the following source code:

```
distance: item1 to: item2
"Calculate the distance between two points"
   ^ item1 point dist: item2 point
```

`item1` and `item2` are the names of the two parameters, which can be chosen freely. To use this script, proceed as in the previous example: select two points in the construction and click somewhere in the sketch to pin the result of the script.

4.1.4 Detailed example with several scripts

In the following section, we present a more complex sketch, integrating a cascading use of scripts to construct the curve of a function and its tangent at a mobile point of the curve.

DRAFT

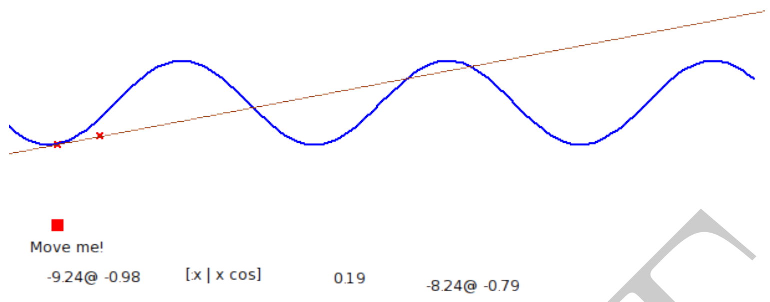


Figure 4.4: Curve and tangent to a point

The final sketch is on the Dr. Geo folder `examples`, it is named `Curve and slope.fgeo`.

In a new empty sketch, we construct a horizontal segment, then add to this segment a free point named "Move me!". This point will be the base to construct the curve as a locus.

Define a function: A script can return any type of object, so that our first construction simply defines a function. To do so, we use a Smalltalk object named block of code (an anonymous function in Scheme and some other languages). We name this script `myFunction`, without parameter:

```
myFunction
"Definition of our function"
   ^ [:x | x cos]
```

Next we insert the function in the sketch[7] So the block of code returned by `myFunction` expects one argument `:x` and it returns the cosine of that argument. We see later how to manipulate this script.

Image of a value by function: Now we calculate the coordinates of a point on the curve. We use our point "Move me!" abscissa and our function previously defined. Our script will have one unique point argument; as we will see we do not need to pass `myFunction` as argument:

```
pointM: item
"Return a point on the curve of myFunction and driven by item point"
   ^ item point x @(self myFunction value: item point x)
```

The point returned by this script has the same abscissa as the argument, and its ordinate is the image of that abscissa by the function.

Note:

- the direct access to the function: `self myFunction`,

[5]To learn about the protocol of this class, write its name in a workspace, select it with the mouse then press the keys CTRL-B. A class browser opens on this class to navigate in its protocol and source code.

[6]As we can see because its name ends in ":"

[7]It is important to add it to the construction to get it included in the file when the sketch is saved.

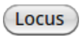
DRAFT

- the way to pass an argument to a block of code, it can be understood as `myFunction(item point x)`.

Now we use this script `pointM:` with the point “Move me!” as argument; the result of this script is of the form `1.2@0.5` representing a pair of coordinates.

With the tool 

, `Points->Coordinates` , we create a point with its coordinates specified by the result of this script.

The locus tool 

, `Lines->Locus` , gives the curve after selecting our two points.

Slope at a point of the function’s curve: We calculate an approximation of the slope at a given point thus:

$$p = \frac{f(x + 0.001) - f(x)}{0.001}$$

This is directly translated in the script `slope:` with a single argument, the point at which to approximate the slope:

```
slope: item
"Compute the slope at the given x point position for myFunction"
| f  x |
   f := self myFunction.
   x := item point x.
   ^ (f value: x + 0.001) - (f value: x) / 0.001
```

Next we insert this script in the construction.
Note:

- The declaration of temporary variables `| f x |`. As explained variables are not typed, so only the name is required.
- A reference to a block of code in a variable `f := self myFunction`. The symbol to assign a value to a variable is “`:=`”.
- The parenthesis! Smalltalk has no concept of operator priority; in fact operators do not exist in this language. The reader is encouraged to study the section about Smalltalk messages in the book *Pharo By Example*.

Calculate and display the tangent to a curve: To construct our tangent we need two points: one on the curve – we have it already – and a second one on the line. For this last one we use the previously calculated slope. Let’s write a second script computing this second point’s coordinates:

```
pointN: item
"Given an initial point, calculate the coordinates of a point on the tangent"
| p x |
   p := self slope: item.
   x := item point x + 1.
   ^ x @ (item point y + p)
```

With the protocol of the class `Point` – message `+` sent to a point – this script can be written in one line:

```
pointN: item
"Given an initial point, calculate the coordinates of a point on the tangent"
   ^ item point + (1@ (self slope: item))
```

DRAFT

We use this script with argument the point on the curve. We get as result the coordinates of the second point on the tangent. With these inputs we construct the second point[8]. The tangent is the line defined by this point and the point on the curve.

When moving the point "Move me!", the tangent is recomputed. Equally interesting: modifying the script `myFunction` updates the whole construction accordingly. A few examples of modifications to this script:

```
^ [:x | x * x / 10]

^ [:x | x cos + (10 * x) sin]

^ [:x | (x * 5) cos + x abs]
```

4.2 Reference methods for Dr. Geo scripts

An argument passed to a script is always a reference to an instance of a class from the hierarchy `DrGMathItem`,the base class of all items used in a construction. So to know about the messages you can send to an argument passed to a script, it is wise to examine the hierarchy of `DrGMathItem`. It contains more than 80 classes, but because of class inheritance, only a few classes are interesting to explore:

- `DrGMathItem`
- `DrGpointItem`
- `DrGDirectionItem` for segment, line, ray, vector
- `DrGArcItem`
- `DrGCircleItem`
- `DrGLocus2ptsItem`
- `DrGPolygonNptsItem`
- `DrGValueItem`

To explore these classes, open a workspace – Ctrl-k after a click in the background of the environment – input and mouse select the class name to explore then press the shortcut Ctrl-b to open a class browser on it.

The following sections contain descriptions of some useful messages, ordered by class.

4.2.1 Math Item

The protocol of the class `DrGMathItem` concerns all types of argument passed to a script.

`<string> item safeName`

`item` : mathematics item
⟶ text representing the item name
Example :

```
name := point1 safeName.
^ name asUppercase.
```

To use this method in a script, create a Dr. Geomath item and give it a name with the style function. Then create the script

[8]Tool `Points->Coordinates`

DRAFT

```
myName: item
^item safeName
```

and use it as usual, clicking on the point and then a place to put the result. Note that changing the name of the item and then forcing an update, for example by moving the item, updates the result of the script.

<boolean> item exist

item : mathematics item
⟶ boolean indicating whether the item exists in the current state of the sketch

Example :
```
line exist ifTrue: [ position := line origin ].
```

<collection> item parents

item : mathematics item
⟶ collection of the parents of this item
Example :
```
point1 := segment parents first.
```

In this example, the parents of a segment joining two points are returned as an array containing the two points. In contrast, the parents of a free point are returned as nil.

item move: vector

item : mathematics item
vector : a vector (x,y) representing the displacement
Example :
`circle move: 2@1.` Move an item in a given direction, while respecting its constraints.

<point> curve closestPointTo: aPoint

curve : mathematics curve
aPoint : coordinates
⟶ coordinates of the point on "curve" the closest to "aPoint".
Example :
```
position := segment closestPointTo: 2@1.
```
Example :
```
position := arc closestPointTo: 2@1.
```

This works when the curve is a line, a segment, a vector, an arc, a circle, a polygon, or a locus.

4.2.2 Point

A point item – point object defined in a Dr. Geo construction – passed as argument to a script is a complex object. It can be a free point on the plane, on a line, an intersection, etc. A few methods can be useful in scripts.

<point> item point

item : point item
⟶ coordinates of this item
Example :
```
abscissa := pointItem point x.
```

DRAFT

For example, with this script

```
showOrdinate: aDrGeoPoint
"ordinate of point"
^ aDrGeoPoint point y
```

The user would invoke this script, pick a point, and pin the result in the sketch showing the ordinate of the selected point.

`item point: aPoint`

`item` : point item
`aPoint` : coordinates
Action : modify the coordinate of "item"
Example :
`pointItem point: 5@2.`

`<float> item abscissa`

`item` : free point on a line
⟶ curvilinear abscissa of this point, in the interval [0 ; 1]
Example :
`a := pointItem abscissa.`

`item abscissa: a`

`item` : free point on a line
`a` : decimal number in the interval [0 ; 1]
Action : modify the curvilinear abscissa of a free point on a line
Example :
`pointItem abscissa: 0.5.`

`item moveAt: aPoint`

`item` : geometric point
`aPoint` : coordinates where to displace "item"
Action : displace "item" to the given position
Example :
`point moveAt: 2@1.`

DRAFT

4.2.3 Straight or curved line

`<float> curve abscissaOf: aPoint`

`curve` : straight or curved line
`aPoint` : coordinates (x,y) of a point
⟶ number in [0 ; 1] curvilinear abscissa of "aPoint" on "curve"
Example :
`a := curve abscissaOf: 2@1.`

`<point> curve pointAt: anAbscissa`

`curve` : straight or curved line
`anAbscissa` : number in [0 ; 1]
⟶ coordinates of a point on "curve" with curvilinear abscissa equal to "anAbscissa"

Example :

```
myPoint := curve pointAt: 0.5.
```

```
<boolean> curve contains: aPoint
```

`curve` : straight or curved line
`aPoint` : coordinates (x, y) of a point
⟶ boolean indicating if the “aPoint” is on “curve”
Example :

```
(curve contains: 0@1) ifTrue: [^ 'Yes!'].
```

4.2.4 Line, ray, segment, vector

```
<point> item origin
```

`item` : line, ray, segment or vector
⟶ coordinates of the origin of this item
Example :

```
item origin.
```

```
<vector> item direction
```

`item` : line, ray, segment or vector
⟶ vector (x, y) indicating the item direction
Example :

```
v := item direction.
slope := v y / v x.
```

```
<vector> item normal
```

`item` : line, ray, segment or vector
⟶ unit vector normal to the item direction
Example :

```
n := item normal.
```

4.2.5 Segment

```
<float> item length
```

`item` : segment
⟶ length of the segment
Example :

```
segment := canvas segment: 0@0 to: 5@5.
l := segment length
```

```
<point> item extremity1
```

`item` : segment
⟶ coordinates of the extremity 1
Example :

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity1.
```

DRAFT

<point> item extremity2

item : segment
⟶ coordinates of the extremity 2
Example :

```
segment := canvas segment: 0@0 to: 5@5.
p := segment extremity2
```

<point> item middle

item : segment
⟶ coordinates of the middle of the segment
Example :

```
segment := canvas segment: 0@0 to: 5@5.
m := segment middle
```

4.2.6 Circle, arc, polygon

<point> item center

item : circle or arc
⟶ coordinates of the centre of the circle or arc
Example :

```
c := item center.
```

<float> item radius

item : circle or arc
⟶ radius of the circle or arc
Example :

```
r := item radius
```

<float> item length

item : circle, arc or polygon
⟶ length of the circle, arc or polygon
Example :

```
l := arc length
```

4.2.7 Value

<float> item valueItem

item : value
⟶ value of this item
Example :

```
n1 := item2 valueItem.
n2 := item2 valueItem.
n1 + n2.
```

item valueItem: v

item : free value item
v : decimal number
Action : modify the value of a free value item

DRAFT

Example :
`item valueItem: 5.2.`

`item position: aPoint`

`item` : value
`aPoint` : coordinates (x, y) of a point
Action : displace at the screen the position of a value item
Example :
`item position: 0.5@2.`

4.2.8 Angle

`<integer> angle degreeAngle`

`angle` : angle, oriented or geometric
⟶ measure of this angle item in degrees
Example :
`angle1 := a1 degreeAngle.`

`<float> angle radianAngle`

`angle` : angle, oriented or geometric
⟶ measure of this angle item in radians
Example :
`angle1 := a1 radianAngle.`

DRAFT

DRAFT

Chapter 5

Dr. Geo Smalltalk sketch

Dr. Geo *Smalltalk sketches* – (DSS) – are sketches entirely defined in the Smalltalk language. This is not about constructing a sketch with the Dr. Geo graphical interface, but about describing a sketch with the Smalltalk language. We provide a nice programming interface with an easy and light syntax.

5.1 Smalltalk sketches by example

Smalltalk itself is a high level language, carefully crafted iteratively for about 10 years at Xerox Parc Research Labs. When a sketch is defined with it, we can use all the power of the language to build a sketch, or to position some objects randomly to get a slightly different sketch at each execution of its Smalltalk code. Therefore, a Smalltalk sketch is freed from the constraints of the graphic user interface while reinforced by the Smalltalk language.

A Smalltalk sketch is source code to execute in a **workspace**, which is a text editor where source code can be written and executed. To open one, use the shortcut Ctrl-k when no window is selected – or click on the background and in the menu select `Tools->Workspace` . It is possible to paste text in with the shortcut Ctrl-v. Read section 7.1.1, p. 75 to learn more about this tool.

Smalltalk sketches are not saved within Dr. Geo. They must be pasted to an external text editor and saved to external files in order to be available for reuse in Dr. Geo. Then the text must be copied from the external file and pasted into a Dr. Geoworkspace in order to be executed again.

To introduce the concepts of Smalltalk sketches, we will study a few examples. Each one will be written in a workspace and its source code will be selected with mouse, then executed with the shortcut Ctrl-d to *Do-it!*[1].

Let's start with the simplest possible Smalltalk sketch:

```
DrGeoCanvas new
```

When executing it, it simply creates a new empty sketch. The Dr. Geo canvas displayed is simplified as there is no toolbar, only the menu bar.

A second example that begins to do something:

```
| c item |
c := DrGeoCanvas new.
item := c point: 1.2 @ -2.
item name: 'A'.
```

[1] Alternatively, this is the entry to select in the contextual menu of the workspace.

DRAFT

Here we define a sketch containing a free point A of coordinates $(1, 2\ ;\ -2)$. An object is added to the construction by sending a message to the canvas, here `#point:` to create a free point given its coordinates. The result is a point object. We can also modify the point by sending a message to it. Here we rename it 'A'.

Let's continue with a third, more interesting example:

```
| c triangle ourRandom m n p |

triangle := [:p1 :p2 :p3 |
c segment: p1 to: p2.
c segment: p2 to: p3.
c segment: p3 to: p1].

ourRandom := [6 - 11 atRandom].

c := DrGeoCanvas new.
m := c point: ourRandom value @ 0.
n := c point: 5 @ 0.
p := c point:  ourRandom value @ 3.
triangle value: m value: n value: p.
```

This example shows us three interesting things:

33. The introduction of more elaborate constructions, not initially implemented in Dr. Geo. Here we define a block of code `triangle` between the square brackets: given three points, construct a triangle. We can compare this to the macro construction method but with a different approach, programming oriented.
34. The definition of a block of code – `ourRandom` – to give us integer random number between -5 and 5. It is used for a random positioning of the points. Therefore each time the sketch is executed, it is slightly different.
35. Assigning the result of a construction – the object we got when sending a construction message to the canvas – is not mandatory. We do it when we need to keep a reference to the constructed object for later use. Here in the block of code `triangle`, we don't keep references to the constructed segments. However, concerning the defined points, we keep references in the local variables `m`, `n` and `p`, for later use as arguments when executing the block of code `triangle`.

To finish with this introduction by example, here is the last one:

```
| c a b d |

c := DrGeoCanvas new.
a := c point: 1@0.
b := c point: 5@0.
d := c line: a to: b.
a color: Color yellow;
   round;
   large.
b hide.
d dashed.
```

Two points and a line are constructed. Then messages are sent to them to modify their styles, including hiding one.

We have finished our small guided tour of Dr. Geo *Smalltalk sketches*. In the following sections, we expose the commands available.

DRAFT

5.2 Reference methods for the Dr. Geo Smalltalk sketch

To add an object to a construction, you send a message to the canvas. The resulting constructed object can be modified as well by sending messages to it. So before adding any object to a canvas, we need to create the canvas with the command `DrGeoCanvas new`.

5.2.1 Various messages

<canvas> DrGeoCanvas new

⟶ Return a canvas and open it. The result is normally assigned to a variable for later use to add constructions.
Example :

```
| canvas |
canvas := DrGeoCanvas new.
```

canvas fullscreen

Action : The sketch is set to fill the Dr. Geowindow.
Example :

```
| canvas |
canvas := DrGeoCanvas new.
canvas fullscreen
```

canvas do: block

`block` : Smalltalk block of code with instructions for construction and/or animation of the sketch.
Action : Execute the block of code in a specific background process. Use it when a construction needs to be done step by step in front of the user or when the sketch is animated.
Example :

```
| canvas point |
canvas := DrGeoCanvas new.
canvas fullscreen.
point := canvas point: 0@0.
canvas do: [
   -5 to: 5 by: 0.1 do: [:x |
      point moveTo: x@(x cos * 3).
      (Delay forMilliseconds: 100) wait.
      canvas update]
]
```

canvas update

Action : Update the canvas after some objects were modified. Used mainly for animation, or for modifications that do not cause an update themselves.

canvas gridOn

Action : Display the grid in the canvas.

DRAFT

canvas centerTo: aPoint

aPoint : Coordinates of a point.
Action : The canvas is displaced so the point given by the argument is at the centre of the canvas window.
Example :
```
canvas centerTo: 5@0.
```

canvas scale: anInteger

anInteger : Scale of the canvas.
Action : modify the scale of the canvas.
Example :
```
canvas scale: 10.
```

Point.

<point> canvas point: aPoint

aPoint : coordinates (x,y).
⟶ free point in the plane with coordinates aPoint.
Example :
```
canvas point: 5@2.
```

<point> canvas pointX: v1 Y: v2

v1 : value object
v2 : value object
⟶ point determined by its coordinates in the form of values placed in the sketch
Example :
```
canvas pointX: (canvas freeValue: 2) hide Y: (canvas freeValue: 5) hide.
```

<point> canvas pointOnCurve: curve at: abscissa

curve : line (straight line, ray, segment, etc.)
abscissa : curvilinear abscissa of the free point, abscissa in interval [0 ; 1]
⟶ free point on a line
Example :
```
myPoint := canvas pointOnCurve: s1 at: 0.5.
```

<point> canvas middleOf: p1 and: p2

p1 : point item or coordinates
p2 : point item or coordinates
⟶ middle of two points
Example :
```
| canvas a i |
canvas := DrGeoCanvas new.
a := canvas point: 1@1.
i := canvas middleOf: a and: 4@4.
```

<point> canvas middleOf: s

s : segment
⟶ middle of a segment

DRAFT

Example :
canvas middleOf: s.

<point> canvas intersectionOf: l1 and: l2

l1 : line
l2 : line
⟶ point of intersection of two lines
Example :
canvas intersectionOf: line1 and: line2

<point> canvas altIntersectionOf: l1 and: l2

l1 : line
l2 : line
⟶ second point of intersection of two lines, when it exists
Example :
canvas altIntersectionOf: line and: circle.

<point> canvas point: block parent: item

block : block of code returning coordinates
item : math item
⟶ pointBlockItem whose coordinates are calculated with the block of code with "item" as argument
Example :

```
| figure s mobile c block |
figure := DrGeoCanvas new.
figure fullscreen.
s:=figure
   segment: (figure point: -5@0)
   to: (figure point: 5@0).
mobile := figure pointOnCurve: s at: 0.1.
block := [:mathItem | |x|
   x := mathItem point x.
   x @ (x * x * x / 25 - x)].
c := figure point: block parent: mobile.
figure locusOf: c when: mobile.
```

<point> canvas point: block parents: itemCollection

block : block of code returning coordinates
itemCollection : math item collection
⟶ point whose coordinates are calculated with the block of code with "itemCollection" as argument
Example :

```
| figure a b d m p |
figure:=DrGeoCanvas new.
a:=figure point: (-2)@1.
b:=figure point: 3@3.
d:=figure line: a to: b.
d color: Color blue.
m:=figure point: 1@(-1).
```

DRAFT

```
p:= figure
   point: [:parents | parents first closestPointTo: parents second point]
   parents: {d . m}.
```

This example uses array notation . to provide the collection needed.

Line.

<line> canvas line: p1 to: p2

p1 : point or coordinates
p2 : point or coordinates
⟶ line passing through these two points
Example :

```
| canvas p1 |
canvas := DrGeoCanvas new.
p1 := canvas point: 0@0.
canvas line: p1 to: 1@2.
```

Note that if one or both arguments are given as literals, those points are not drawn, and are not available in the sketch to click and drag. The direction of the line thus cannot be changed interactively.

<line> canvas parallel: d at: p

d : direction (line, segment, vector,...)
p : point or coordinates
⟶ line parallel to direction d and passing through point p
Example :

```
| canvas a d |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
d := canvas line: (-2)@1 to: 3@3.
canvas parallel: d at: a.
```

<line> canvas perpendicular: d at: p

p : point or coordinates
d : direction (line, segment, vector, ...)
⟶ line perpendicular to direction d and passing through point p
Example :

```
canvas perpendicular: d at: 1@5.
```

<line> canvas perpendicularBisector: s

s : segment
⟶ perpendicular bisector to s
Example :

```
canvas perpendicularBisector: (canvas segment: 0@0 to: 4@4)
```

<line> canvas perpendicularBisector: a to: b

a : point or coordinates
b : point or coordinates
⟶ perpendicular bisector to segment joining a and b

DRAFT

Example :
```
canvas perpendicularBisector: 0@0 to: 4@4
```

```
<line> canvas angleBisector: a
```
`a` : geometric angle defined by **three points**
⟶ angle bisector of the angle `a`
Example :
```
canvas angleBisector: angle
```

```
<line> canvas angleBisectorSummit: a side1: b side2: c
```
`a,b,c` : points defining a geometric angle $\widehat{bac}$
⟶ angle bisector of the angle $\widehat{bac}$
Example :
```
canvas angleBisectorSummit: 0@0 side1: 1@0 side2: 0@1
```

Ray.

```
<ray> canvas ray: o to: p
```
`o` : point or coordinate, the origin
`p` : point or coordinates, point anywhere on the ray
⟶ ray defined by its origin and a second point
Example :
```
| canvas a |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
canvas ray: 0@0 to: a.
```

Segment.

<segment> canvas segment: p1 to: p2
`p1` : points or coordinates
`p2` : points or coordinates
⟶ segment defined by two points
Example :
```
| canvas a |
canvas := DrGeoCanvas new.
a := canvas point: 5@5.
canvas segment: 10@10 to: a.
```

Circle.

```
<circle> canvas circleCenter: c to: p
```
`c` : point or coordinates, centre of the circle
`p` : point or coordinates, point on the circle
⟶ circle defined by its centre and a point
Example :

DRAFT

```
| canvas a |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
canvas circleCenter: a to: 10@4.
```

<circle> canvas circleCenter: c radius: r

c : point or coordinates, centre of the circle
r : numeric item or numeric value, radius
$\longrightarrow$ circle defined by its centre and radius

Example :

```
| canvas a r |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
r := canvas freeValue: 4.
canvas circleCenter: a radius: r.
canvas circleCenter: 4@4 radius: 5
```

Arc.

<arc> canvas arc: p1 to: p2 to: p3

p1 : point or coordinates, 1^{ere} extremity of the arc
p2 : point or coordinates representing a point on arc
p3 : point or coordinates, 2^{eme} extremity of the arc
$\longrightarrow$ arc defined by its extremities and a point

Example :

```
| canvas a b |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
b := canvas point: 0@5.
canvas arc: a to: b to: -1 @ -2.
```

<arc> canvas arcCenter: O from: A to: B

O : point or coordinates, centre of the arc
A : point or coordinates, origin of the arc
B : point or coordinates, so the angle is $\widehat{AOB}$
$\longrightarrow$ arc defined by its centre and angle $\widehat{AOB}$

Example :

```
| canvas a b |
canvas := DrGeoCanvas new.
a := canvas point: 1@5.
b := canvas point: 0@5.
canvas arcCenter: a from: b to: -1 @ -2.
```

Polygon.

<polygon> canvas polygon: collection

collection : collection of points or coordinates; summits of the polygon
$\longrightarrow$ polygon defined by its summits

Example :

DRAFT

```
| canvas b |
canvas := DrGeoCanvas new.
b := canvas point: 1@3.
canvas polygon: {1@2. b. 0@0. d.}.
```

<polygon> canvas regularPolygonCenter: center vertex: summit sides: number

`center` : point or coordinates, centre of the polygon
`summit` : point or coordinate, a summit of the polygon
`number` : value item or numeric value, number of summits of the polygon
⟶ regular polygon defined by its centre, one summit and number of summits
Example :

```
| canvas b |
canvas := DrGeoCanvas new.
b := canvas point: 1@3.
canvas regularPolygonCenter: b vertex: 1@1 sides: 7.
```

Geometric transformations.
Geometric transformations are to transform any kind of geometric objects: point, segment, line, ray, vector, circle, arc and polygon.

<transformed item> canvas rotate: item center: centre angle: angle

`item` : point, segment, line, ray, vector, circle, arc, polygon
`item` : object to transform
`centre` : point or coordinates, rotation centre
`angle` : value item or numeric value, rotation angle
⟶ transformed object
Example :

```
| canvas c k l |
canvas := DrGeoCanvas new.
c := canvas point: 5@5.
k := 3.14159.
l := canvas line: 0@0 to: 5@5.
canvas rotate: l center: c angle: k.
canvas rotate: l center: 0@0 angle: Float pi / 3.
```

<transformed item> canvas scale: item center: centre factor: k

`item` : point, segment, line, ray, vector, circle, arc, polygon
`item` : object to transform
`centre` : point or coordinates, homothety centre
`k` : value item or numeric value, homothety factor
⟶ transformed object
Example :

```
| canvas c k l |
canvas := DrGeoCanvas new.
c := canvas point: 0@5.
k := -3.
l := canvas line: 0@0 to: 5@5.
canvas scale: l center: c factor: k.
canvas scale: l center: 0@4 factor: 5.
```

DRAFT

```
<transformed item> canvas symmetry: item center: centre
```

`item` : point, segment, line, ray, vector, circle, arc, polygon
`centre` : point or coordinates, symmetry centre
⟶ transformed object
Example :

```
| canvas a |
canvas := DrGeoCanvas new.
a := canvas point: 4@2.
canvas symmetry: a center: 0@0.
```

```
<transformed item> reflect: item axe: axis)
```

`item` : point, segment, line, ray, vector, circle, arc, polygon
`axe` : line, symmetry axis
⟶ transformed object
Example :

```
| canvas a d polygon |
canvas := DrGeoCanvas new.
a := canvas polygon: 0@0 . 2@0 . 3@2 . 0@4.
d := canvas line: (-1)@(-1) to: 4@(-1).
canvas reflect: a axis: d.
```

```
<transformed item> canvas translate: item vector: vector
```

`item` : point, segment, line, ray, vector, circle, arc, polygon
`vector` : vector or coordinates
⟶ transformed object
Example :

```
| canvas u a|
canvas := DrGeoCanvas new.
u := canvas vector: (canvas point: 1@1) to: (canvas point: 3@2).
a := canvas translate: (canvas point: 2@1) vector: u.
```

Example :

```
| canvas u a|
canvas := DrGeoCanvas new.
a := canvas translate: (canvas point: 2@1) vector: 2@1.
```

Locus of a point.

```
<locus> canvas locusOf: c when: m
```

`m` : mobile point on a line
`c` : fixed point depending on the mobile point `m`
⟶ locus
Example :

```
canvas locusOf: c when: mobile.
```

Vector.

```
<vector> canvas vector: o to: e
```

`o` : point or coordinates, vector origin
`e` : point or coordinates, vector extremity

DRAFT

⟶ vector
Example :
```
| canvas b |
canvas := DrGeoCanvas new.
b := canvas point: 0@5.
canvas vector: b to: -1 @ -2.
```

`<vector> canvas vector: p`

p : point or coordinates, the vector coordinates
⟶ vector from 0@0 to point
Example :
```
| canvas p |
canvas := DrGeoCanvas new.
p := canvas point: 5@5.
canvas vector: p.
canvas vector: -5 @ -5
```

Number.

When applying a method to a canvas that creates a numeric value, the value is placed as a Dr. Geomath item somewhat at random in the sketch, and is initially hidden. It can be revealed with the show command described below.

`<point> pointItem coordinates`

pointItem : point
⟶ coordinates (static) of pointItem
Example :
```
| canvas c p segment |
canvas := DrGeoCanvas new.
segment := canvas segment: 0@0 to: 3@5
p := canvas pointOnCurve: segment at: 0.5.
c := p coordinates.
canvas freeValue: (c x)
```

`<value> canvas abscissaOf: pointOrVector`

pointOrVector : point or vector item
⟶ abscissa (dynamic) item of `pointOrVector`
Example :
```
| canvas m x |
canvas := DrGeoCanvas new.
m := canvas middleOf: 10@5 and: 7@8.
x := canvas abscissaOf: m.
```

The abscissa of a point is its x coordinate. The abscissa of a vector is the difference between the x coordinates of its initial and final points.

`<value> canvas ordinateOf: pointOrVector`

pointOrVector : point or vector item
⟶ ordinate (dynamic) item of `pointOrVector`
Example :
```
| canvas m x |
canvas := DrGeoCanvas new.
```

DRAFT

```
m := canvas middleOf: 10@5 and: 7@8.
x := canvas ordinateOf: m
```

The ordinate of a point is its y coordinate. The ordinate of a vector is the difference between the y coordinates of its initial and final points.

```
<value> canvas freeValue: v
```

`v` : initial value
⟶ free value item, initially hidden, randomly placed
Example :

```
v:= canvas freeValue: (-1 arcCos).
v show
```

```
<value> canvas lengthOf: segment | circle | arc | vector)
```

`segment|circle|arc|vector` : segment, circle, arc or vector
⟶ number item, the item length
Example :

```
canvas lengthOf: v1.
```

```
<value> canvas distance: line | point to: point2
```

`line|point` : line or point
`point2` : point
⟶ number, distance between two points or a point and a line
Example :

```
canvas distance: l1 to: a.
```

```
<value> canvas slopeOf: line
```

`line` : line
⟶ number, slope of the line
Example :

```
| canvas p |
canvas := DrGeoCanvas new.
p := canvas slopeOf: d.
```

Dr. Geodoes not handle infinite slope for vertical lines well. If a slope is placed in a sketch, and then the line is made vertical, the slope disappears from the sketch, and its value does not update in the data pane.

Angle.

```
<value> canvas angle: a to: b to: c
```

`a` : point
`b` : point, angle summit
`c` : point
⟶ geometric angle $\widehat{abc}$ in the interval $[0 ; \pi]$
Example :

```
| canvas a b c angle |
canvas := DrGeoCanvas new.
a :=canvas point: 0@0.
b :=canvas point: 0@4.
c :=canvas point: 3@0.
```

DRAFT

```
angle := canvas angle: b to: a to: c.
angle show.
```

`<value> canvas angle: v1 to: v2`

`v1` : vector
`v2` : vector
$\longrightarrow$ oriented angle given two vectors in the interval $]-\pi\ ;\ \pi]$
Example :

```
| canvas v1 v2 a b c angle |
canvas := DrGeoCanvas new.
a :=canvas point: 0@0.
b :=canvas point: 0@4.
c :=canvas point: 3@0.
v1 := canvas vector: a to: b.
v2 := canvas vector: a to: c.
angle := canvas angle: v2 to: v1.
angle show.
```

Equation.

`<equation> canvas equationOf: lineOrCircle`

`lineOrCircle` : line or circle
$\longrightarrow$ equation of the line or circle
Example :

```
| canvas e d |
canvas := DrGeoCanvas new.
d := canvas line: 0@0 to: 15@13.
e := canvas equationOf: d. e show.
```

5.2.2 Modification of object attributes

To modify object attributes, we send messages to the objects. So the attributes are always modified after creating the objects.

`item color: aColor`

`item` : any object in the canvas
`aColor` : a `Color`, see methods in this class for existing colors : `Color black, Color red, Color blue, Color orange, Color yellow,...`
Action : modify the item color
Example :

```
pointA color: Color green.
```

`item name: aString`

`aString` : a string, text
Action : rename item
Example :

```
segment name: '[AB]'.
```

`item hide`

`item` : any item

Action : hide an item
Example :
`point hide`

`item show`
`item` : any item
Action : show an item
Example :
`value show`

`line small`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : Set thickness of line to small
Example :
`circle small.`

`line normal`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : Set thickness of line to normal
Example :
`arc normal.`

`line large`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : Set thickness of line to large
Example :
`polygon large.`

`line plain`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : plain, undotted, style line
Example :
`polygon plain.`

`line dashed`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : dash style line
Example :
`polygon dashed.`

`line dotted`
`line` : line item (straight line, ray, circle, lieu, etc.)
Action : dotted style line
Example :
`arc dotted.`

`point cross`
`point` : point item

DRAFT

Action : cross shape for point
Example :
`a cross.`

`point round`
`point` : point item
Action : round shape for point
Example :
`a round.`

`point square`
`point` : point item
Action : square shape for point
Example :
`a square.`

`point small`
`point` : point item
Action : small sized point
Example :
`a small.`

`point medium`
`point` : point item
Action : medium sized point
Example :
`a medium.`

`point large`
`point` : point
Action : large sized point
Example :
`a large.`

`item moveTo: point`
`item` : point or value item
`point` : coordinates
Action : move the item to the given position, if its constraints permit
Example :

```
| canvas a |
canvas := DrGeoCanvas new.
a := canvas point: 0@0.
a moveTo: 5@5.
canvas update
```

DRAFT

5.2.3 Complementary methods

The `DrGeoCanvas` class proposes in the category `helpers` complementary methods to ease the computation of complex, interactive sketches.

```
canvas plot: aBlock from: x0 to: x1
```

`aBlock` : block of code, with one argument, to describe a function on an interval
`x0` : number, lower boundary of the function's domain
`x1` : number, upper boundary of the function's domain
Action : display the curve representing the function described by the block of code, in the interval [x0 ; x1], and display a free point on the x axis together with a relative point on the curve, representing the value of the function at that point.
Example :

```
canvas plot: [:x| x * x] from: -3 to: 3.
```

```
<block> canvas float: float1 at: aPoint from: float2 to: float3 name: aStri
```

`float1` : initial value
`aPoint` : left position of the ruler
`float2` : minimum value
`float3` : maximum value
`aString` : name of the value
$\longrightarrow$ block of code returning the current value of the ruler
Action : construct a ruler at the given position with a **decimal** value in the interval [float2 ; float3]
Example :

```
| canvas A F |
canvas := DrGeoCanvas new.
A := canvas float: 1 at: -10@4 from: 0 to: 10 name: 'A'.
F := canvas integer: 3 at: -10@3 from: 0 to: 10 name: 'F' showValue:
true.
A value + F value.
```

There are other variants, some for integer values.

5.3 Gallery of examples

To illustrate the use of Dr. Geo Smalltalk sketches, we present below a small set of examples. It shows you some possibilities that we hope will inspire you for your own needs. For each example for which we give the Smalltalk source code, we encourage you to copy and paste it into a Dr. Geo workspace and run the code.

5.3.1 Animate a figure

These examples rely on time handling and background processes.

We begin with a simple animation in order to understand the concept:

```
| figure p pause |\\
figure:=DrGeoCanvas new.\\
p := figure point: 0@0.\\
pause := Delay forSeconds: 0.2.\\
figure do: [
   100 timesRepeat: [
      p mathItem moveTo: (p mathItem point + (0.1@0)).\\
      figure update.\\
```

DRAFT

```
      pause wait]].
```

The Delay class supports setting time intervals for pausing Dr. Geo. The do: and timeRepeat: control structures repeat a code block that moves its argument item across the sketch. The update is necessary so that the point is displayed in its new position after each move.

A second example with a more elaborate sketch:

```
| figure s r u pause |
figure := DrGeoCanvas new fullscreen.
s := figure segment: 0@ -1 to: 4@ -1.
r := figure pointOnCurve: s at: 0.8.
s := figure segment: 0@0 to: 0@1.
u := figure pointOnCurve: s at: 0.7.
u round small; color: Color blue.
1 to: 100 do: [:n|
   u := figure
      point: [:parents| |y t|
         y := parents first point y.
         t := parents second point x.
         (n / 5) @ t * y * (1 - y)]
      parents:  {u . r}.
   u round small; color: Color blue].
pause := Delay forSeconds: 0.1.
figure do: [
   0 to: 1 by: 0.05 do: [:x |
      r mathItem setCurveAbscissa: x.
      figure update.
      pause wait]].
```

5.3.2 Sierpinski triangle

This example largely relies on a recursive block of code for drawing triangles and subdividing them to a specified number of levels.

```
| triangle c |
triangle := [:s1 :s2 :s3 :n |
    c segment: s1 to: s2;
        segment: s2 to: s3;
        segment: s3 to: s1.
    n > 0 ifTrue:
        [triangle
            value: s1
            value: (c middleOf: s1 and: s2) hide
            value: (c middleOf: s1 and: s3) hide
            value: n-1.
        triangle
            value: (c middleOf: s1 and: s2) hide
            value: s2
            value: (c middleOf: s2 and: s3) hide
            value: n-1.
        triangle
            value: (c middleOf: s1 and: s3) hide
            value: (c middleOf: s2 and: s3) hide
            value: s3
```

DRAFT

```
            value: n-1.]].

c := DrGeoCanvas new.
triangle
    value: (c point: 0 @ 3)
    value: (c point: 4 @ -3)
    value: (c point: -4 @ -3)
    value: 3.
```

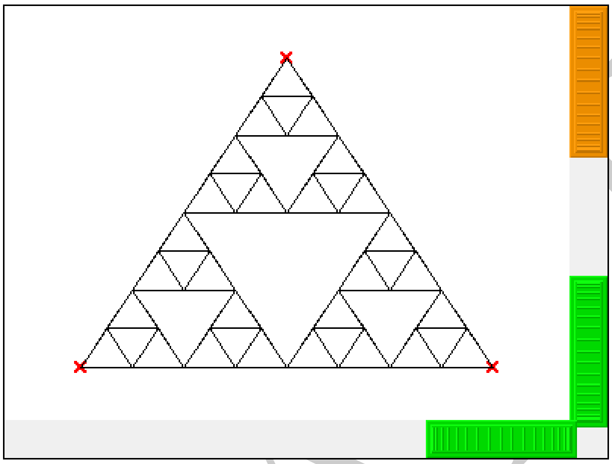


Figure 5.1: Triangle de Sierpinski

DRAFT

Part III

Applications

DRAFT

DRAFT

Chapter 6

Didactic applications

This chapter is an aid to studying Dr. Geo using didactic examples. Although in the previous chapters several examples were presented, here the approach is little more concrete, while still very original. This chapter was created using contributions from various cultural sources.

6.1 Perimeter and area

One possible didactic use of Dr. Geo is through its Smalltalk script system to resolve classic geometry exercises.

As an example, we will show the solution of the following classic problem – involving the Pythagorean theorem:

> *A right trapezoid ABCD where the bases and height are known. Calculate its perimeter and area.*

Solution:
First we construct the Dr. Geo sketch as follows:

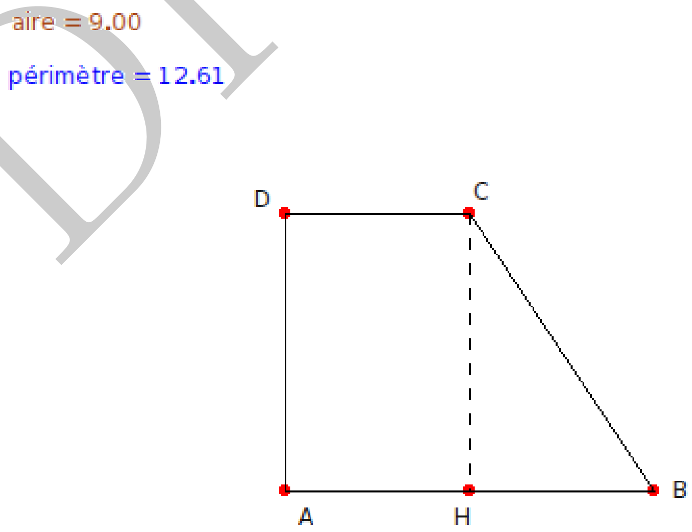


Figure 6.1: Right trapezoid

The sketch contains the data to use to resolve the problem. First we can answer the question of the area. To do so we write a Smalltalk script with arguments the two bases and the height:

```
areaTrapezoidBase1: b1 base2: b2 height: h
"Calculate the area of a right trapezoid given
its bases and one height"
   ^ h length * (b1 length + b2 length) / 2
```

To calculate the perimeter, we write a script where we calculate the length of BC with the Pythagorean theorem:

```
perimeterTrapezoidBase1: b1 base2: b2 height: h
"Calculate the area of a right trapezoid given
its two bases and one height"
| hb bc |
   hb := (b2 length - b1 length) abs.
   bc := (hb squared + h length squared) sqrt.
   ^ b1 length + b2 length + h length + bc
```

It is not difficult, if you follow the same model, to develop other similar examples.

6.2 Theorem and conjectures

With Smalltalk scripts one can solve exercises, but can also understand theorems more deeply and verify conjectures. In this section we start to analyse Ptolemy's theorem:

> *Given an inscribed quadrilateral, the sum of the products of its opposite sides is equal to the product of its diagonals.*

We construct the sketch as below where we implemented two scripts to calculate respectively the sum of the products of its opposite sides and the product of its diagonals.

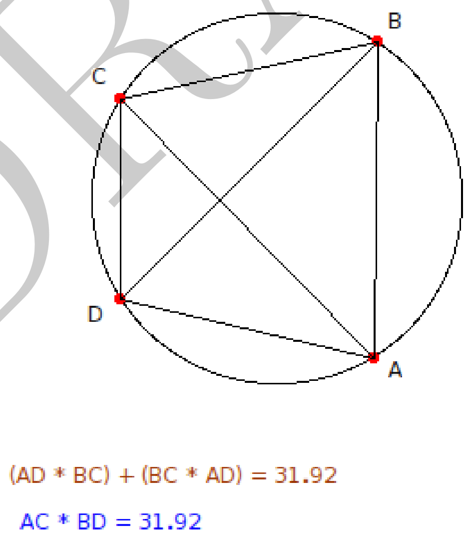


Figure 6.2: Ptolemy's theorem: inscribed quadrilateral

The first script:

```
ptolemySumS1: ab s2: bc s3: cd s4: ad
"Select the four consecutive sides of the quadrilateral, then the location for the result"
^ (ad length * bc length) + (ab length * cd length)
```

The second script:

DRAFT

```
ptolemyProductD1: ac d2: bd
"Select the two diagonals of the quadrilateral, then the location for the result"
^ ac length * bd length
```

As we can see, the values returned by the scripts, as stated by Ptolemy's theorem, are equal[1] When we dynamically modify the sketch, the script values are always equal, except in the following situation where the quadrilateral is not convex:

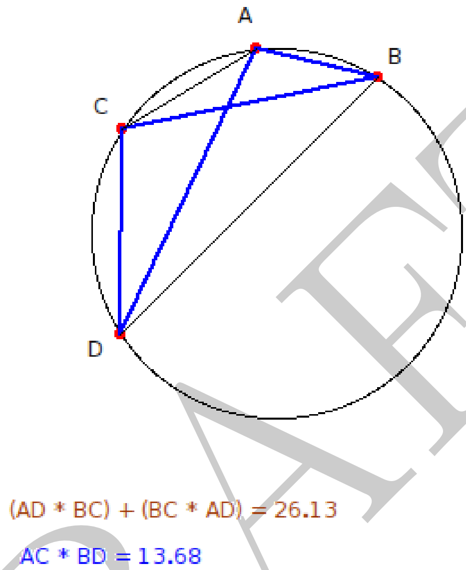


Figure 6.3: Ptolemy's theorem: non convex inscribed quadrilateral

In this case the theorem is not true and the previous text of the theorem is inaccurate. It should be reformulated as follow:

> *Given an inscribed CONVEX quadrilateral, the sum of the products of its opposite sides is equal to the product of its diagonals.*

Now a additional conjecture can be stated: is the theorem still valid when the convex quadrilateral is **non inscribed**?

With Dr. Geo we verify this conjecture is false with the following counterexample bellow. To build this counterexample, we have just detached the point B from the circle by dragging it with the touch $\boxed{\text{SHIFT}}$ pressed at the same time.

The reader will easily use Dr. Geo to construct additional didactic examples, perhaps more famous, relative to the Pythagorean and Euclidean theorems.

[1]This is only a numerical example, not a proof.

DRAFT

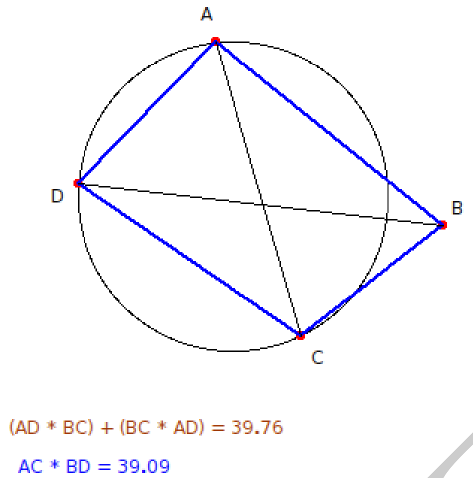


Figure 6.4: Counterexample of the conjecture

6.3 Irrational numbers

A classic construction of irrational numbers, known as the Teodoro spiral, gives the geometric construction of integer square roots. It begins with an isosceles right triangle.

Let's start with the triangle OAB where $OA = AB = 1$:

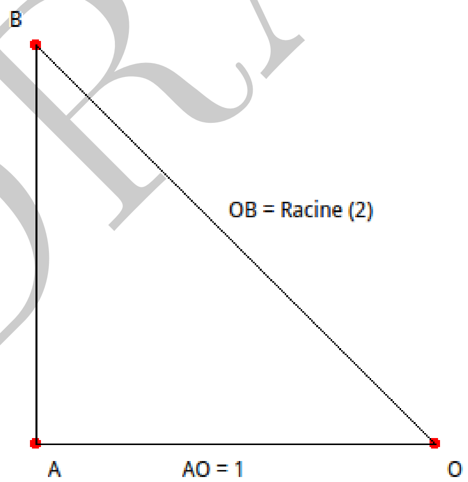


Figure 6.5: Construction of the square root of 2

Using the Pythagorean theorem we have OB equal to the square root of 2. Now, with the sketch, we construct another right triangle with the sides OB and BC so that $BC = 1$.

Still using the Pythagorean theorem, it is clear the hypotenuse OC of OBC has a length equal to the square root of 3. The process can be repeated without limit to get the square roots of all positive integers.

The iterative nature of this construction can be naturally represented in a Smalltalk sketch.

DRAFT

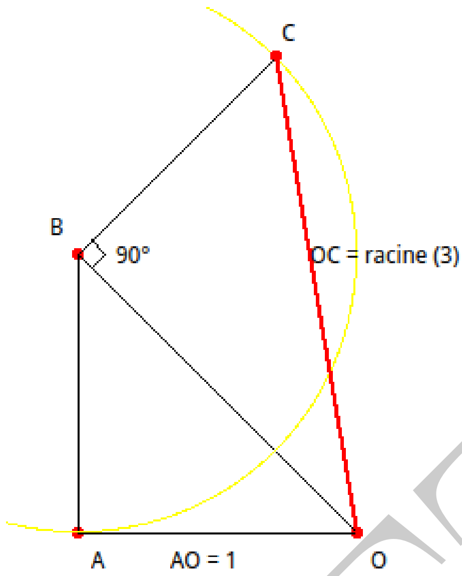


Figure 6.6: Construction of the square root of 3

Let's consider the following Smalltalk sketch source code:

```
| sketch triangle |
sketch := DrGeoCanvas new fullscreen.
triangle := [:p1 :p2 :p3  :n |
   |s1 s2 s3 perp circle p4 |
   s1 := sketch segment: p1 to: p2.
   s2 := (sketch segment: p2 to: p3) color: Color red.
   s3 := sketch segment: p3 to: p1.
   perp := (sketch perpendicular: s3 at: p3) hide.
   circle := (sketch circleCenter: p3 to: p2) hide.
   p4 := (sketch altIntersectionOf: circle and:  perp) hide.
   n > 0 ifTrue: [triangle value: p1 value: p3 value: p4 value: n -1]].

triangle
   value: 0@0
   value: -1@0
   value: -1@1
   value: 50
```

The triangle at the beginning is defined by the coordinates of its vertices. The source code is a direct transcription of the logic of the construction, integrating its iterative nature with a recursive loop. We use the `#hide` message several times to mask intermediate constructions, in order not to overload the user's perception with background constructions. Once executed, DR. GEO gives the following sketch.

The hypotenuse length of each triangle is the square root of an integer in the interval [2 ; 52].

DRAFT

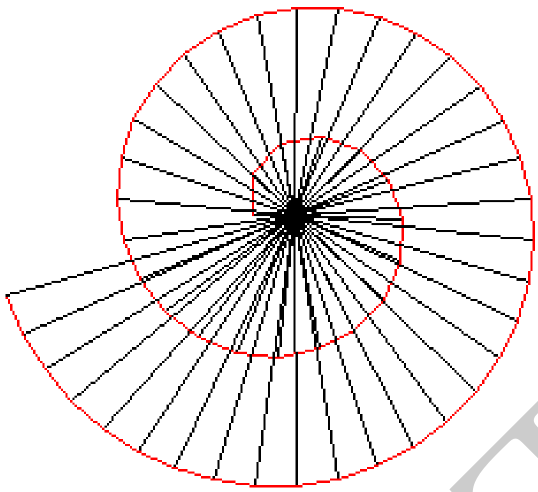


Figure 6.7: Teodoro spiral

The same spiral with intermediate items shown exposes how difficult it will be to *hand construct* such a sketch:

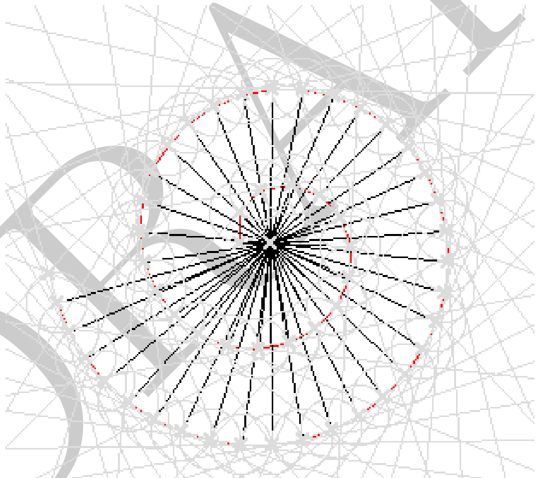


Figure 6.8: Teodoro spiral with hidden item revealed

As a bonus, here is an animated version of the previous Smalltalk sketch. To do this we send to the canvas the messages `#do:` and `#update`. They are documented in the chapter 4, p. 27.

Note the use of the class `Delay` to slow down the construction:

```
| sketch triangle delay|
sketch := DrGeoCanvas new fullscreen.
triangle := [:p1 :p2 :p3  :n |
   |s1 s2 s3 perp circle p4 |
   s1 := sketch segment: p1 to: p2.
   s2 := (sketch segment: p2 to: p3) color: Color red.
   s3 := sketch segment: p3 to: p1.
   perp := sketch perpendicular: s3 at: p3.
   circle := sketch circleCenter: p3 to: p2.
```

DRAFT

```
   p4 := sketch altIntersectionOf: circle and:  perp.
   sketch update.
   (Delay forMilliseconds: 200) wait.
   perp hide. circle hide. p4 hide.
   n > 0 ifTrue: [triangle value: p1 value: p3 value: p4 value: n -1]].

sketch do: [triangle value: 0@0 value: -1@0 value: -1@1 value: 50]
```

6.4 Baravelle spiral

As we saw previously, in Smalltalk sketches it is easy to construct, intuitively and simply, sketches *to visualise* recursive or iterative situations in programming.

We can go one step further by modifying the previous Smalltalk code – used to construct irrational numbers – to get a famous sketch of the mathematics literature: the Baravelle spiral constructed from similar equilateral right triangles.

The code to construct this spiral is as follows:

```
| sketch triangle |
sketch := DrGeoCanvas new fullscreen.
triangle := [:p1 :p2 :p3  :n | |s1 s2 s3 m  perp circle p4 |
   s1 := sketch segment: p1 to: p2.
   s2 := sketch segment: p2 to: p3.
   s3 := sketch segment: p3 to: p1.
   m := (sketch middleOf: p1 and: p3) hide.
   perp := (sketch perpendicular: s3 at: p3) hide.
   circle := (sketch circleCenter: p3 radius: (sketch distance: m to:  p3) hide) hide.
   p4 := (sketch altIntersectionOf: circle and:  perp) hide.
   n > 0 ifTrue: [triangle value: m value: p3 value: p4 value: n -1]].

triangle
   value: (sketch point: 0@5)
   value: (sketch point: 5@5)
   value: (sketch point: 5@0)
   value: 9.
triangle
   value: (sketch point: 0@ -5)
   value: (sketch point: -5@ -5)
   value: (sketch point: -5@0)
   value: 9
```

With this sketch and the corresponding Smalltalk code we clearly perceive the recursive nature of the construction mechanism. An interesting problem for the reader is to establish when the two branches of the spiral converge.

DRAFT

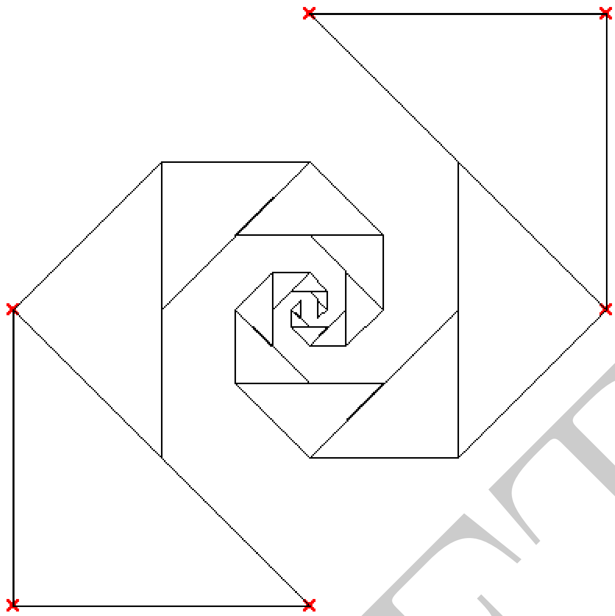


Figure 6.9: The Baravelle spiral after executing the Smalltalk code

6.5 Pappus Chain

Another classic use of a Smalltalk programmed sketch is based on its numeric ability to reproduce a geometric sketch knowing its analytic characteristics.

The example we propose is the famous “Pappus Chain”.

The successive circles’ centres and radii are analytically known, and in fact rational. It is therefore easy to reproduce this sketch by programming.

```
| sketch circle a o m|
sketch := DrGeoCanvas new fullscreen.
circle := [:n | |r c p |
   r := (sketch freeValue: 15 / (n squared + 6)) hide.
   c := sketch point: (15 / (n squared + 6) * 5) @ (15 / (n squared + 6) * n * 2).
   c small; round.
   p := sketch circleCenter: c radius: r.
   n > 0 ifTrue: [circle value: n - 1]].

circle value: 10 .
a := (sketch point: 5@0) name: 'A'.
o := (sketch point: 0@0) name: 'O'.
m := sketch
   middleOf: o
   and: ((sketch point: 15@0) name: 'B').
m name: 'M'.
sketch
   circleCenter: m to: o;
   circleCenter: a to: o;
   line: a to: o.
```

The source code is relatively intuitive and it does not require any comment.

A non-trivial exercise for the reader consists of determining a ruler and compass construction for the figure.

DRAFT

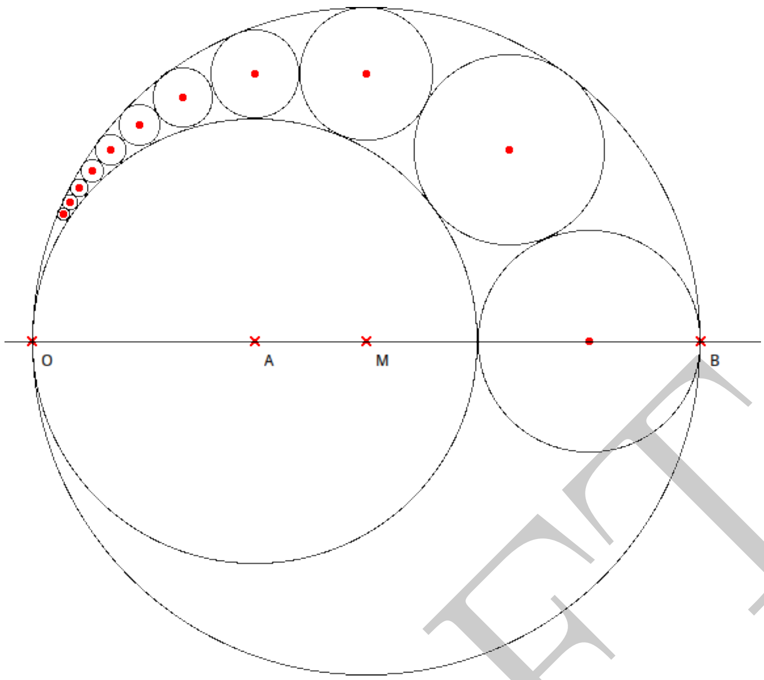


Figure 6.10: Catena di Pappo

6.6 π calculus

The approximation of π played an important role in the history of mathematics. Numerous methods were proposed, offering a variety of improvements over time, from elementary geometry to calculus. We will examine one of the earliest approaches to the problem, originally carried out by Archimedes, called – although this phrase is not precisely accurate – the **method of exhaustion**. This approach has however the advantage of exposing the essence of the methodology.

Archimedes described a simple ruler and compass construction of regular polygons inscribed inside and outside a circle, and laborious calculations of the lengths of their sides, doubling the number of sides at each step up to 96. In modern terms, this gives three digits of precision, approximately 3.14. We, however, can proceed much more simply, letting Dr. GEOdo the hard work, and continuing to polygons of more than a million sides, which takes us nearly to the limits of ordinary computer arithmetic, that is, 16 decimal places for floating point numbers (3.141592653589793). Using much more advanced methods on supercomputers has made it possible to calculate ten trillion digits of π.

We start with the construction of an inscribed regular hexagon. Create a numeric value, and set it to 6. Create a segment (the diameter) and bisect it to get the center. Create a circle with the given center through one of the endpoints of the segment. Create a regular polygon with the given center, one of the endpoints as a vertex, and the numeric value as the number of sides.

The idea of the of the exhaustion method starts with approximating the length of the circle with the perimeter of the P_0 hexagon; then to calculate an approximation of π as the quotient of the perimeter of that hexagon by the diameter of the circle. Clearly, since the side of the hexagon is the same length as the radius of the circle, the resulting initial approximation of π is 3.

In a second step, we construct an inscribed dodecagon. To construct the previous regular hexagon, we used the Dr. Geo tool to construct a regular polygon with 6 vertices. To construct the dodecagon, we just need to change this free value item to 12, and the polygon

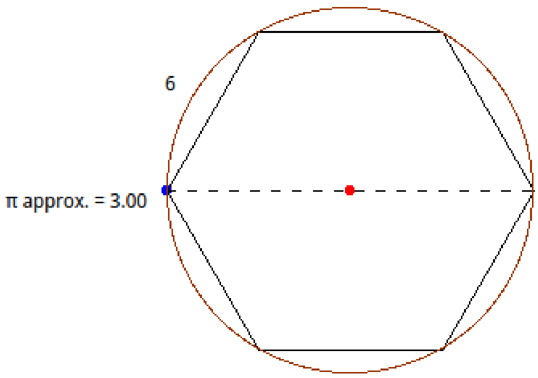


Figure 6.11: Inscribed regular hexagon

will update automatically. We calculate the P_1 perimeter, then an approximation of π as the quotient of the perimeter by the diameter of the circle.

A small script is written to calculate this quotient; its arguments are the polygon and the circle diameter:

```
approxPIpolygon: poly diameter: d
"PI approximation given an inscribed regular polygon.
Arguments: the polygon and the circle diameter"
   ^ poly length / d length
```

Now apply this script to the constructed polygon and its diameter (the segment we started with), and pin the result in the sketch.

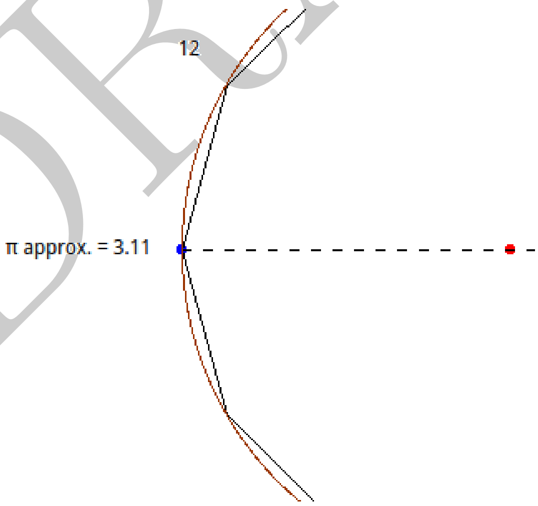


Figure 6.12: π approximation

When increasing the number of sides of the polygon, by editing the free value, we improve the approximation.

To display both the approximation of π and its accuracy, we can modify the script to display them together:

```
approxPIpolygon: poly diameter: d
"PI approximation given an inscribed regular polygon.
```

DRAFT

```
Arguments: the polygon and the circle diameter"
^ Array
   with: poly length / d length
   with: (poly length / d length - Float pi) abs
```

As we increase the number of sides of the polygon, how quickly does this process approach the true value? How far can we go?

DRAFT

DRAFT

Chapter 7

Various tips

Due to Dr. Geo integration with the Pharo Smalltalk environment, there are a few genies we can invoke. Most of them are hidden to the user. It is not that we want to restrain their use, but that we don't want to overload the user when discovering Dr. Geo. As we will show in the following sections, these genies can be invoked from menus, keyboard shortcuts or Smalltalk code.

7.1 Programming

In this section we present a few tools to use when writing Smalltalk scripts or composing sketches, beginning with the workspace, the debugger, and the inspector.

7.1.1 Workspace

To show a workspace, click on the Dr. Geo environment background then do Ctrl-k[1].

A workspace, at first glance, is like a text editor. But it is in fact a console to edit Smalltalk code: to write it, to compile it, and to execute it. It is of course possible to paste in code copied from somewhere else.

After invoking a workspace, paste the source code of the Smalltalk sketch below[2] :

```
| sketch function p integral summits |

function := [:x | x * x ].
summits := OrderedCollection new.
sketch := DrGeoCanvas new.
p := sketch point: -1@0.
p hide.
summits add: p.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].

p := sketch point: 1@0.
summits add: p hide.

integral := sketch polygon: summits.
integral color: Color blue.
```

[1]Depending one your system, replace Ctrl by Alt.

[2]To paste a text, try with the shortcut Ctrl-v or from its contextual menu (right click).

DRAFT

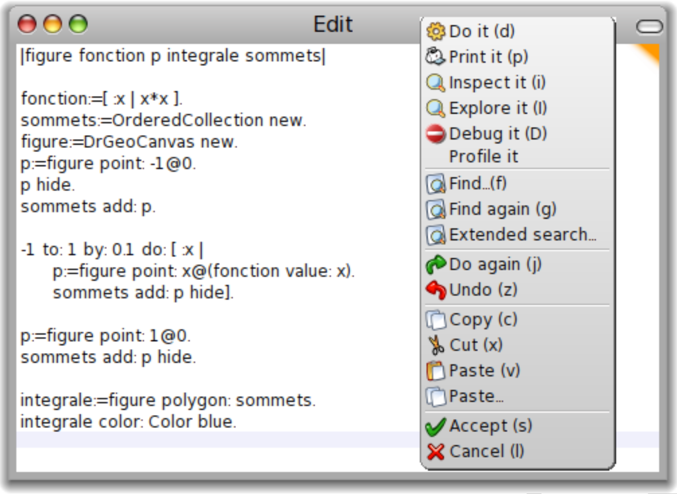


Figure 7.1: Your workspace with the pasted source code and its contextual menu

To compile and to execute this source code, just select it with the mouse and invoke *Do it(d)* in the contextual menu. These two operations can also be done at the keyboard: CTRL-A to select all the source code then CTRL-D to execute it. You immediately get the result of this code, an interactive programmed sketch.

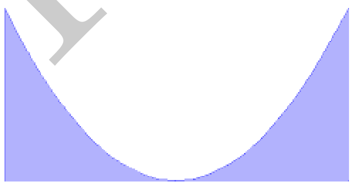


Figure 7.2: Result after executing the source code: the integral of the function in [-1 ; 1]

When executing complex Smalltalk code, running it with a profiling option let you see its bottlenecks. To do so, in the contextual menu invoke the command *Profile it*. The source code is executed, the sketch is built, and then a profile window informs you about the execution time in different part of the code and the various invoked methods. It is a wonderful way to navigate the execution tree of the code and look for methods consuming too many cycles.

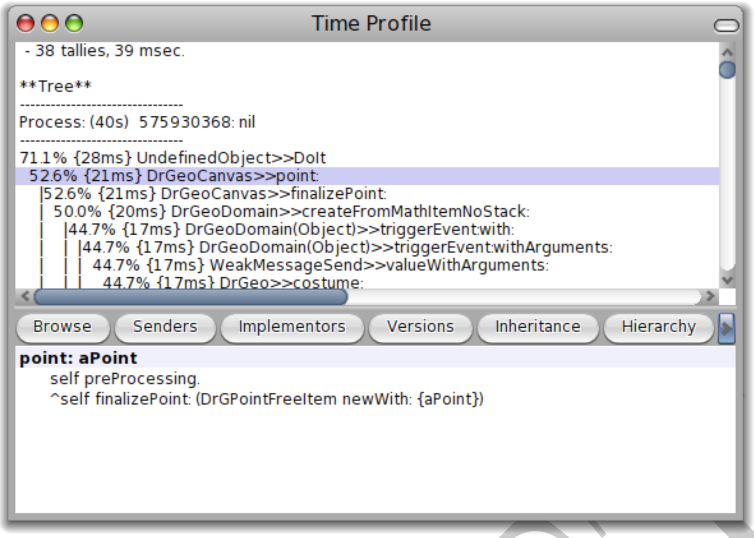


Figure 7.3: Dr. Geo profiling

Last refinement: stepping through code. This is done through the debugger. Select the code to debug, then in the contextual (right-click) menu, choose the command *Debug it*. The debugger is invoked on the first line of the selected source code, which is executed step by step with the button *Over*, one method call at a time. In the bottom area of the debugger window, at the right, the local variables are shown with type information. Other buttons allow other refinements in step by step execution. You should explore them before looking at Smalltalk documentation.

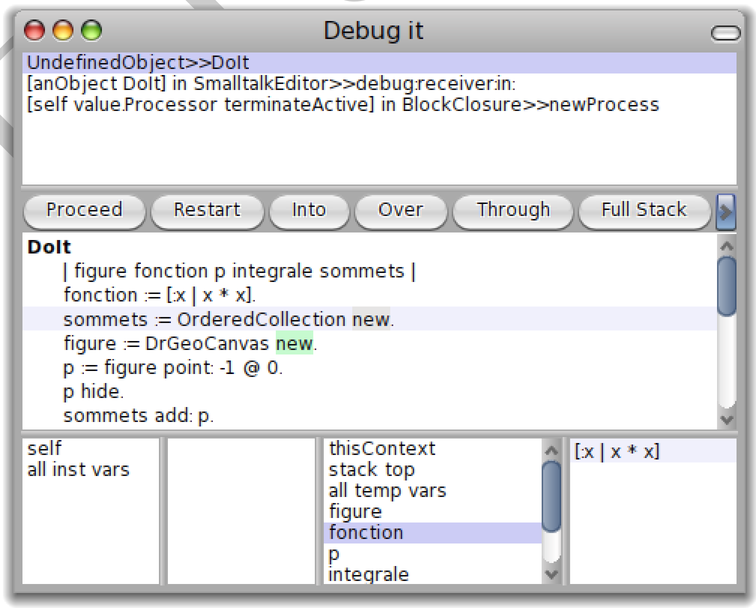


Figure 7.4: The Dr. Geo debugger

DRAFT

7.1.2 Debugger

As shown in the previous section, the debugger lets you execute the code step by step. In the Dr. Geo environment you can invoke the debugger on any selected code in a workspace with the keyboard shortcut Alt-D.

Moreover, the debugger can be invoked in the source code by adding a line `self halt.` – like a breakpoint. In our previous example we modify the source code as follow:

```
...
p := sketch point: -1@0.
p hide.
summits add: p.

self halt.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].
...
```

7.1.3 Inspector

In the inspector dialogue, the user examines the attributes of an instance or a variable. The information in the view is updated automatically whenever the inspected object changes.

In our previous example, suppose we want to see the contents of the `summits` collection. In this case, we very simply add a line of code to send the message `inspect` to `summits`. The place in the code where we put it is not very important[3]. It can be at the beginning or the end because we do not have a breakpoint or step by step execution:

```
...
p := sketch point: -1@0.
p hide.
summits add: p.

summits inspect.

-1 to: 1 by: 0.1 do: [:x |
   p := sketch point: x @ (function value: x).
   summits add: p hide].
...
```

[3]For obvious reasons, we must avoid adding it in the loop.

DRAFT

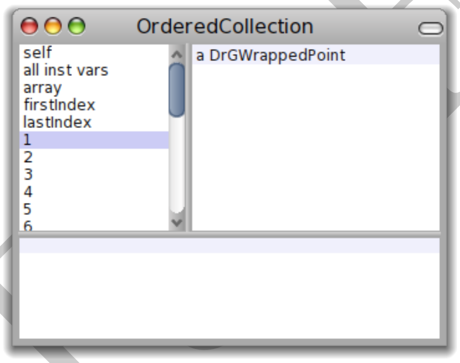


Figure 7.5: The inspector on the variable `summits`

DRAFT

DRAFT

Index

DRAFT

DRAFT

DRAFT